# MASSIVELY PARALLEL BAYESIAN OBJECT RECOGNITION

**by**

**Isidore Rigoutsos**

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Computer Science Department

Courant Institute of Mathematical Sciences

New York University

August 1992

Approved: _____

Professor Robert A. Hummel

Faculty Advisor

# Report Documentation Page

*Form Approved*
*OMB No. 0704-0188*



| 1. REPORT DATE | 2. REPORT TYPE | 3. DATES COVERED |
|---|---|---|
| **AUG 1992** | | **00-00-1992 to 00-00-1992** |

| | |
|---|---|
| 4. TITLE AND SUBTITLE<br>**Massively Parallel Bayesian Object Recognition** | 5a. CONTRACT NUMBER |
| | 5b. GRANT NUMBER |
| | 5c. PROGRAM ELEMENT NUMBER |
| 6. AUTHOR(S) | 5d. PROJECT NUMBER |
| | 5e. TASK NUMBER |
| | 5f. WORK UNIT NUMBER |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)<br>**New York University,Department of Computer Science,70 Washington Sq South,New York City,NY,10012** | 8. PERFORMING ORGANIZATION REPORT NUMBER |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | 10. SPONSOR/MONITOR'S ACRONYM(S) |
| | 11. SPONSOR/MONITOR'S REPORT NUMBER(S) |

12. DISTRIBUTION/AVAILABILITY STATEMENT
**Approved for public release; distribution unlimited**

13. SUPPLEMENTARY NOTES

14. ABSTRACT

**The problem of model-based object recognition is a fundamental one in the eld of computer vision, and represents a promising direction for practical appli- cations. We describe the design, analysis, implementation and testing of a system that employs geometric hashing techniques, and can recognize three-dimensional objects from two-dimensional grayscale images. We examine the exploitation of parallelism in object recognition, and analyze the performance and sensitivity of the geometric hashing method in the presence of noise. We also present a Bayesian interpretation of the geometric hashing approach. Two parallel algorithms are outlined: one algorithm is designed for an SIMD hypercube-based machine whereas the other algorithm is more general, and relies on data broadcast capabilities. The rst of the two algorithms regards geometric hashing as a connectionist algorithm. The second algorithm is inspired by the method of inverse indexing for data retrieval. We also determine the expected distribution of computed invariants over the hash space: formulas for the distributions of invariants are derived for the cases of rigid, similarity and a ne transformations, and for two di erent distributions (Gaussian and Uniform over a disc) of point features. Formulas describing the de- pendency of the geometric invariants on Gaussian positional error are also derived for the similarity and a ne transformation cases. Finally, we present an interpretation of geometric hashing that allows the geometric hashing algorithm to be viewed as a Bayesian approach to model-based object recognition. This interpretation is a new form of Bayesian-based model matching, and leads to natural, well-justi ed formulas. The interpretation also provides a precise weighted-voting method for the evidence-gathering phase of geometric hashing. A prototype object recognition system using these ideas has been implemented on a CM-2 Connection Machine. The system is scalable and can recognize aircraft and automobile models subjected to 2D rotation, translation, and scale changes in real-world digital imagery. This system is the rst of its kind that is scalable uses large databases, can handle noisy input data, works rapidly on an existing parallel architecture, and exhibits excellent performance with real world, natural scenes.**

| 15. SUBJECT TERMS | | | | | |
|---|---|---|---|---|---|
| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
| a. REPORT<br>**unclassified** | b. ABSTRACT<br>**unclassified** | c. THIS PAGE<br>**unclassified** | **Same as Report (SAR)** | **221** | |

**Standard Form 298 (Rev. 8-98)**
Prescribed by ANSI Std Z39-18

And then the man steps right up to the microphone
And says at last as the time bell rings
‘‘Thank you good night now it’s time to go home’’
and he makes it fast with one more thing
‘‘We are the Sultans of Swing’’

Mark Knopfler

Γιά τούς γονεῖς μου

**Acknowledgments**

My most sincere thanks go to my advisor Bob Hummel for his support, guidance and encouragement during the work on this dissertation. To him, I also owe the understanding of how to identify promising research directions.

I am very grateful to Haim Wolfson of the Tel Aviv University, School of Mathematical Sciences for being a constant source of enthusiasm, criticism and insight.

My thanks also go to Stephane Mallat for support, and guidance, and for insight on the "tricks of the trade."

During my years of research, Ruud Bolle, Andrea Califano and Rakesh Mohan of IBM's T. J. Watson Research Center provided stimulating feedback and criticism, for which I sincerely thank them.

I am also grateful to Alan Mainwaring and Lewis Tucker of Thinking Machines Corporation for all the lengthy discussions we had during my first steps in the exciting world of data parallel computing.

Thanks are also due to Marsha Berger for providing, through the adaptive mesh refinement work, a source of support and interesting distraction from my thesis research during my summer months in the Courant Institute.

The work in this dissertation would not have been possible without access to a Connection Machine, through the DARPA Connection Machine Network Server Program. I especially thank the University of Maryland UMIACS center for generously providing access to their resources.

Work on this dissertation has been supported by an IBM Graduate Research Fellowship, and by AFAL contract F33615-89-1087.

**Abstract**

The problem of model-based object recognition is a fundamental one in the field of computer vision, and represents a promising direction for practical applications.

We describe the design, analysis, implementation and testing of a system that employs geometric hashing techniques, and can recognize three-dimensional objects from two-dimensional grayscale images. We examine the exploitation of parallelism in object recognition, and analyze the performance and sensitivity of the geometric hashing method in the presence of noise. We also present a Bayesian interpretation of the geometric hashing approach.

Two parallel algorithms are outlined: one algorithm is designed for an SIMD hypercube-based machine whereas the other algorithm is more general, and relies on data broadcast capabilities. The first of the two algorithms regards geometric hashing as a connectionist algorithm. The second algorithm is inspired by the method of inverse indexing for data retrieval.

We also determine the expected distribution of computed invariants over the hash space: formulas for the distributions of invariants are derived for the cases of rigid, similarity and affine transformations, and for two different distributions (Gaussian and Uniform over a disc) of point features. Formulas describing the dependency of the geometric invariants on Gaussian positional error are also derived for the similarity and affine transformation cases.

Finally, we present an interpretation of geometric hashing that allows the geometric hashing algorithm to be viewed as a Bayesian approach to model-based object recognition. This interpretation is a new form of Bayesian-based model matching, and leads to natural, well-justified formulas. The interpretation also

provides a precise weighted-voting method for the evidence-gathering phase of geometric hashing.

A prototype object recognition system using these ideas has been implemented on a $CM$-2 Connection Machine. The system is scalable and can recognize aircraft and automobile models subjected to $2D$ rotation, translation, and scale changes in real-world digital imagery. This system is the first of its kind that is scalable, uses large databases, can handle noisy input data, works rapidly on an existing parallel architecture, and exhibits excellent performance with real world, natural scenes.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

This dissertation addresses the problem of model-based object recognition of three-dimensional objects in two-dimensional grayscale images. In particular, we describe the design, analysis, implementation and testing of a recognition system that can identify three-dimensional objects from real world grayscale photographs using a database of stored models. The models in the image can be rigid-, similarity-, or affine-transformed versions of prototype models in the database.

By *object recognition* we mean,

- the recovery of the imaged object's identity, and
- the recovery of the transformation that the model has undergone.

The problem of object recognition is a fundamental one in the fields of computer vision and robotics. Perhaps the most promising research direction in image analysis, and the one most likely to lead to industrial and commercial applications, is the area of object recognition where the search is confined within a finite set of observable models.

## 1.1 Object Recognition: the four stages

In all of the object recognition systems, one can distinguish four stages: data acquisition, feature extraction, matching, and verification. We examine each of these stages in more detail.

### 1.1.1 Data Acquisition

Data acquisition is carried out via the use of "sensors." Sensors are devices sensitive to a variety of modalities. Commonly used sensors are sensitive to one of the following: X-rays, visible spectrum light, infrared, microwaves, and ultrasound. Sensors can be either active, in which case they emit an energy beam and subsequently record the signal that is returned from the scene objects, or passive.

The output of a sensor is typically a discrete valued function and results from sampling the input signal at regularly spaced intervals.[1] The spatial pattern of the sample points is called a "tessellation." For two-dimensional signals, the rectangular tessellation has been used in the overwhelming majority of systems. Recently, a prototype light sensor based on an hexagonal tessellation has made its appearance [96].

Independent of whether the tessellation is rectangular, hexagonal, or triangular, *regularly* tessellated sensors operating at rates of 30 frames/sec produce large amounts of data that typically cannot be processed and analyzed in real time with today's available computing power. In order to accommodate the problem, fovea-like sensors have also been suggested [9]. The resolution of a foveated sensor

[1] Actually, the recorded value is not the value of the function at the location where the sampling takes place, but rather the function's *integral* over a very small area of the sensor.

is not constant across the field of view, but instead decreases as one moves from the center of the field to the periphery, similarly to the human fovea [63]. An immediate result of such a tessellation is the considerable reduction of the amount of data that needs to be processed, to the point that real-time vision tasks might be feasible. However, such a tessellation also necessitates the development of new image processing algorithms, a not-so-straightforward task [100].

In what follows, we will concern ourselves with sensors that are sensitive to the visible spectrum light, and have a rectangular tessellation. The output of such sensors is a $2D$, grayscale (or color) intensity image.

### 1.1.2 Feature Extraction

When presented with an intensity image, an object recognition system's task is to identify and locate the object(s) present in the scene that generated the image. The first step toward this goal is the reduction of the amount of the input data.

Given that the object, or objects, of interest occupy only a small portion of the viewed scene, most of the data present in the sensor's output is extraneous and not relevant to the task at hand. Feature extraction attempts to identify *interesting* pieces of the input signal.

It is not easy to define what consists a feature. The definition of a feature is directly related to the model representation, i.e. the way each of the recognizable by the system objects is represented and stored in the computer. Such a representation is in turn related to the way a model can appear in the context of the sensor data [11], and is application specific. Traditionally, the following image characteristics have been used as features: linear and curvilinear segments, curvature extrema, curvature discontinuities, conics etc.

Due to the importance of this stage's output, the problem of feature extraction has received a great lot of attention over the years; in particular, the topic of edge detection (i.e. extraction of linear and curvilinear segments from gray-level intensity data) has attracted the attention of a large number of researchers. This interest was the result of experimental evidence attesting to the importance of boundaries for the human visual system [2]. A large number of techniques and algorithms were developed for feature extraction. However, a presentation of these techniques escapes the scope of this dissertation, and the reader is referred to one of the relevant textbooks such as [7,43,63].

Various features can be combined together to generate object descriptions. The way the various features are combined depends on the application and/or the method. In a number of approaches, geometric information is derived from the extracted features: for example, the position of a certain feature, the curvature of a constant-curvature curvilinear segment, the eccentricity of a conic, etc. This information is used in the representation of the corresponding object. Also, relational information, e.g. relative distance or relative orientation between features, has proven useful in object recognition.

Before we conclude this section, we mention an important issue pertaining to feature extraction, that of sensor noise. Indeed, the sensing devices are not perfect, but instead introduce "measurement" and "amplification stage" noise. This noise manifests itself as small random perturbations of the values of the sampled modality and can potentially cause problems during the feature extraction process. Consequently, a preprocessing or "filtering" stage typically precedes the feature-extraction stage; the goal of the filtering stage is the reduction of the random perturbations introduced during the sampling.

### 1.1.3 Matching

Once the set of models that are to be recognizable by the system has been chosen, the form of representation of the models must be fixed. As already noted, the representation of the models dictates the feature types that the feature-extraction stage will attempt to detect in the sensory output.

Once the types of features have been determined, a database is built containing information about those features that can be identified in objects from the set of recognizable models. The task of the matching stage is to identify the model, or models, whose features approximately match a (sub-)set of the features generated by the feature-extraction module.

The matching stage is the most crucial component of an object recognition system. A number of techniques have been developed toward this end. But in all cases there is a trade-off between the reliability and the computation cost: techniques that produce very reliable results are computationally heavy, and vice versa. Most matching algorithms have been based on cross-correlation techniques, tree- or graph-search, clustering, or indexing: the technique generally depends on the feature type.

The matching technique should allow for partial occlusion, rotation, translation, and scale changes, as well as for small amounts of data perturbation. The output of the matching stage is a set of hypotheses regarding the identity of the models that are embedded in the scene. Sometimes, a measure of belief can be associated at this stage with each of the models in the set; this measure allows the hypotheses' relative ranking. Together with the set of models, the matching stage also recovers the transformation that the corresponding model is assumed

to have undergone.

Several matching techniques can be viewed as "filters" or "sieves" that considerably reduce the number of candidate hypotheses as to the identity of the object(s) in the scene and the transformation the objects have undergone. In this dissertation, we will concentrate on the geometric hashing approach to matching [45,59,61,62]. This approach uses geometric invariants to represent the various models. Geometric invariants are also used to index into the database of models (see also section 2.2.1).

### 1.1.4 Verification

The output of the matching module is a set of hypotheses regarding the identity of the object or objects embedded in the scene. These hypotheses are subsequently piped into the verification stage whose purpose is to evaluate the quality of the hypotheses and either accept or reject them.

In order to evaluate the quality of the hypotheses that are generated, the vision system projects the candidate models onto the scene and the fraction of the model accounted for by the available input data is computed. "Optimal" cutoff values (thresholds) are typically pre-determined either empirically or in an *ad hoc* fashion. These thresholds can be either model-dependent or constant across the various models, and their use allows the verification module to decide whether to accept or reject certain hypotheses.

The majority of vision systems use empirically-determined thresholds very successfully. In some cases, and under certain simplifying assumptions, a *theoretical* analysis makes possible the computation of threshold values that are a function of the scene and complexity of the model [37]. However, to our knowledge, no

current operational system makes use of theoretically-determined thresholds.

## 1.2 The Scope of this Dissertation

In this dissertation, we concentrate on four different issues:

- Exploitation of parallelism for performing object recognition;
- Analysis of the expected probability distributions of invariants over the hash space for a number of transformation and model feature distribution combinations;
- Analysis of how sensor noise propagates through the stage of invariant computations; and
- Formulation of a Bayesian interpretation of model matching with geometric hashing.

### 1.2.1 Exploitation of Parallelism

In chapter 3, we present two data-parallel algorithms for performing geometric hashing.

• The first algorithm is designed for an SIMD hypercube-based parallel architecture; the algorithm has a "connectionist" flavor with information flowing via communication patterns.

• The second algorithm is more general, based on data broadcast capabilities, and suitable for any type of parallel architecture.

In addition to these two algorithms, we also present a novel *radix-sort* algorithm for SIMD hypercube-based architectures.

### 1.2.2 Distributions of Invariants

In chapter 4, we derive precise as well as approximate formulas and qualitative results for the statistical distribution of geometric invariants. The results are derived for a number of transformation (rigid, similarity, affine) and feature-distribution (uniform, Gaussian) combinations.

The analysis corroborates that the non-uniform distribution of invariants over hash space is endemic to all indexing-based approaches to model based object recognition. In chapter 5, we show how one can use the knowledge of the index distributions to develop techniques that result in much faster implementations of indexing-based object recognition methods.

### 1.2.3 Modeling of Noise

In chapter 6, we study the behavior of geometric hashing techniques in the presence of noise.

We show that, under the assumption that the noise introduced by the sensor and the feature-extraction module can be modeled as a Gaussian random process, the computed indices follow a Gaussian distribution to a first order approximation.

We also perform the noise analysis for the similarity and affine transformations, and show that the effect of noise in the latter case is more pronounced.

### 1.2.4 Bayesian Interpretation

In chapter 7, we present an interpretation of geometric hashing which shows that the algorithm can be viewed as a Bayesian, maximum-likelihood object recognition method; the hypotheses span the discrete collection of models and the discrete pairings of image features to model features.

We make use of the results from chapters 4 and 6 to show how an adaptive weighted-voting scheme can be used to accumulate evidence for model/basis tuples in the geometric hashing framework.

The validity of our theory is demonstrated in chapter 8 where we describe a complete object recognition system that makes use of the ideas. The system is implemented on a SIMD hypercube-based parallel machine (a Thinking Machines Corporation $CM$-2) and can recognize objects that have undergone a similarity transformation, from a library containing the models of 14 aircraft and 18 production automobiles. The system is tested using real-world imagery, works rapidly, and exhibits exceptional performance.

# Chapter 2

# An Introduction to Model-based Object Recognition

Most of the successful object recognition systems have been model-based. In these systems, the search is confined to a finite set of observable models. A priori information about these *recognizable* models is maintained in an appropriately structured database. The type of information contained in the database depends on the scheme by which models are represented, and also on the type of features (e.g. points, edges, conics, etc.). Apart from having given rise to a number of successful systems, the *model-based vision* paradigm offers the possibility of a well-defined analyzable formulation.

During the matching stage, the task of a model-based vision system is to determine the model identity (equivalently: a set of model features), and a transformation for each model that is present in the scene. The transformation brings the set of model features in correspondence with a (possibly proper) subset of image features. During the search for the model identity and the appropriate transformation, the system has access to the information that is stored in the

system's database. Things may be complicated if the objects that are contained in the input image are partially occluded, and the attributes of the extracted features (e.g. position, orientation, etc.) are corrupted by noise.

In general, object recognition involves some type of search. The search can take place over the set of extracted features and recognizable models: in this case, the system attempts to determine *correspondences* between (sub-)sets of model and image features; these correspondences are consistent with the permissible transformations. Since the number of all possible "pairings" between model and image feature sets is exponential in the cardinality of the feature sets, straightforward implementations result in an unfavorable time complexity. In an attempt to efficiently prune the tree search, a number of systems have made use of local constraints with considerable success.

Alternatively, the search can take place in the space of transformations: in this case, the system attempts to determine a *transformation* that brings (sub-)sets of model and image features in correspondence.

These two approaches, namely the search over the space of extracted features and the search over the space of transformations, represent the main paradigms that have dominated the field of object recognition in the last ten years.

## 2.1 A Survey of the Object Recognition Field

In this section, we briefly describe some of the most representative object recognition systems that have been developed during the last two decades.

One of the earliest object recognition systems was developed by Roberts [82]. The system was able to recognize convex polyhedral objects under the weak perspective transformation. It controlled and pruned the search by considering only

vertices that were connected by an edge, and thus could not handle occlusion. Unlike Roberts' system, the models in ACRONYM [19] were generalized cylinders. ACRONYM used symbolic constraints to control and effectively prune the search, and could handle both noise and occlusion.

A related approach to that of ACRONYM's was taken in Goad's system [35]. The system used quantitative (as opposed to symbolic) constraints to control the search. Goad's system also introduced the notion of the two stage (off-line stage, on-line stage) recognition algorithm, where data precomputed during a first phase (off-line) are used during the phase of actual recognition (on-line) in order to speed up the processing.

The use of geometric constraints (such as distance and angle) as an efficient way for pruning the search while matching image and model features, was advocated by Bolles in his LFF and 3DPO systems [15,16]; LFF is used to recognize $2D$ objects from intensity images, whereas 3DPO is used for recognition of $3D$ objects from range data. Geometric constraints are also used in the RAF system of Grimson [38,39]. In Grimson's system, the search is structured around an interpretation tree, and exhibits exponential time complexity if the input image (intensity data) includes spurious data. More recently, the BONSAI system [31] exploits *unary* and *binary* constraints to control the search of the interpretation tree, and prune the search space: the input to BONSAI comprises range images of parts that have been designed using a CAD tool.

The HYPER system of Ayache and Faugeras [3] also belongs to the category of systems that attempt to determine correspondences between sets of model and image features. HYPER is used to recognize $2D$ objects from intensity images. However, its success is dependent on the quality of the polygonal approximations

of the input image's contours. Furthermore, it is sensitive to noise and does not deal with the occlusion of edges.

Lowe's system, SCERPO [68], is a complete object recognition system that recognizes polyhedral $3D$ objects from intensity images, under the perspective transformation. SCERPO attempts to reduce the complexity of the search by performing perceptual groupings of image features. However, it typically deals with only one or two models in the model database at any given time.

More recently, work by Kak [53] shows that efficient algorithms can prove beneficial in reducing the complexity in the case of systems that search over the sets of features. In particular, Kak uses bipartite matching in conjunction with the notion of discrete relaxation to perform recognition of $3D$ objects using the output of a structured-light scanner.

In addition, a number of other systems have been developed that search the space of allowed transformations. The classic representative of this approach is the generalized Hough transform [5,8,91]: the method is a generalization of the Hough transform [44] and is used to detect arbitrary shapes. In the generalized Hough transform framework, the recognition of objects is achieved by recovering the transformation that brings a large number of model features in correspondence with image features. The transformation is described in terms of a set of transformation parameters, and votes for these parameters are accumulated by hypothesizing matchings between subsets of model and image features. The generalized Hough transform requires the quantization of a range of values for each of the parameters, thus resulting in decreased accuracy. The space requirements are exponential in the number of the parameters.

The system by Mundy and Thompson [69,70] uses large Hough tables to per-

form recognition of $3D$ objects from $2D$ input data, under the weak perspective transformation. To constrain the space of possible transformations, the system uses the notion of the "vertex-pair." A vertex pair consists of two vertices and the two edges forming one of the vertices. An improved version of Mundy and Thompson's system [85] remedies the problem of fixed parameter quantization by iteratively refining the quantization around volumes of interest (histogram peaks), until the required precision was achieved. A similar system is the one of Linnainmaa [64] which introduced the notion of the "triangle-pair;" triplets of vertices from the image are matched against triplets of model vertices in order to hypothesize a transformation under the perspective projection model; however, the system provides multiple alternatives that require examination.

The approaches of the last three systems can be considered as special cases of a more general scheme called "*alignment*" [97]. In alignment, one seeks a model from the model database together with a transformation from the allowed class of transformations such that the object being viewed and the transformed model are in correspondence; for those transformations where the number of corresponding features exceeds a certain threshold, a verification procedure is invoked. Another alignment-based system, RANSAC [30], is used to recognize objects under perspective transformation, for a known camera position. Huttenlocher's ORA system [50] on the other hand, performs recognition assuming the weak perspective transformation model. More recently, alignment ideas have been combined with efficient string matching in order to perform *unoccluded* polygonal object recognition [83].

The approach of Ullman and Basri [98] is considerably different. The basic idea here is that each *topologically different* model view can be expressed as a

linear combination of a small number of $2D$ views of the model. The method assumes that the transformation of the model to the scene can be modeled by an orthographic projection, and can handle $3D$ rigid as well as non-rigid transformations of the models. Scene clutter causes considerable problems. The scheme has been treated mostly theoretically. Some preliminary results indicate reasonable performance with databases containing a handful of objects.

Another general scheme that also involves search over the space of transformations is the geometric hashing scheme. Although based on the same geometric principles as alignment, geometric hashing differs from alignment in the algorithmic approach. Since this dissertation revolves around the geometric hashing scheme, we chose to survey the object recognition systems that have been based on geometric hashing ideas in a separate section (section 2.3).

All of the systems that have been described so far, as well as those based on the geometric hashing scheme, typically represent the database models using a small number of homogeneous, local features. These features "define" the objects. Furthermore, the objects under consideration are treated in isolation from the rest of the scene. Unlike these systems, CONDOR [92] is the first system that takes the approach of performing context recognition first, and then instantiates the individual components. Natural objects such as *sky*, *ground*, *foliage* are included in the system's vocabulary. A special-purpose database contains all the necessary information about the world. The introduction of context results in increased flexibility at the expense of a major increase of the computational complexity. The input to the system can be any combination of intensity, range, color or other data modalities. The output is a labeled $3D$ model of the input image, with the labels referring to the object classes that can be recognized by the system.

The system by Swain [93] can recognize deformable objects and substances described by mass nouns by making use of color information. Thus it is similar in flavor to CONDOR. However, its use of precomputed invariants for the different database models in a two-stage algorithm, brings the system closer to the ones that are based on hashing/indexing ideas (see section 2.3). Notably, Swain's system can recognize objects independent of background and viewpoint variations, occlusion, scale, and lighting conditions.

Vayda and Kak's INGEN system [99] performs object classification based on the overall shape of the object. For certain object recognition tasks it suffices to categorize the objects based on their general shape (e.g. parallelepiped, cylinder, etc.) and independently of their size. Because of the large variations in size, feature-based object recognition techniques are not easily applicable. INGEN uses a *hypothesize-and-verify* approach to determine the pose and generic shape of objects from range data. Hypotheses generated for each region in the segmented range data can be combined using either information contained in a combinability graph, or proximity and continuity heuristics. Use of the combinability graphs controls the combinatorial explosion by efficient pruning of the search space.

Another system with no knowledge of a *geometric* or *structural* model for each of the database objects is the one by Stark and Bowyer [88]. In their system, object classes are described in terms of the functional properties shared by all the $3D$ objects in the class. The various functional properties are represented using procedural knowledge. The system has been successfully tested with a database of 100 objects belonging to the "chair" class; the output of a CAD tool was used to provide the test input to the system.

Dickinson [26] presents yet another approach to $3D$ object recognition from

intensity images. His system uses a small set of volumetric primitives which can be assembled to form the objects that can be recognized. An important component to the system is a hierarchy of $2D$ features (such as contours, faces, groups of faces) that are generated by projecting the primitives based on a set of viewer-centered orientations; conditional probabilities capture the relation between nodes at different levels of the hierarchy, and can be computed off-line. The number of orientations is fixed and thus independent of the number of models the system can recognize. During recognition, the system uses a bottom-up approach and precomputed conditional probabilities to extract primitives in the input image, as well as the connectivities of the primitives. Using the primitive and connectivity information, the system indexes into the model database to recover the identity of the viewed object. Bergevin's PARVO system [10] takes a similar approach to that of Dickinson's but makes use of "geons" [12] as the modeling primitives.

Kriegman and Ponce's approach [57] also exploits the relation between the shape of intensity image contours and the models of $3D$ objects of revolution. Under the assumption that the image contours are the projections of either surface discontinuities or occluding contours, Kriegman and Ponce use *elimination theory* to construct the implicit equations of the contours under perspective and weak perspective projections; the equations are parameterized by the object's position and orientation. In the current system, edge segments are grouped into contours manually, and no extraneous data are fed into the algorithm. Also, contours that are neither surface nor occluding discontinuities are manually removed. Although they obtain good results, the success of the approach relies heavily on the quality of segmentation.

## 2.2 Indexing Methods

The geometric hashing/indexing methods represent an alternative, efficient and highly parallelizable approach to performing pattern matching. These methods borrow from the technique of "hashing," a fundamental technique in the field of computer science [1,56].

The basic idea behind hashing is the partitioning of a possibly *infinite* data set, $\mathcal{D}$, into a *finite* set of groups $G_i$, $i = 1, 2, 3, \ldots, g$. The $G_i$'s form the *hash table* data structure. A *hash* function, $h$, whose domain is the set from where the members of $\mathcal{D}$ obtain values, and range the set $\{1, 2, 3, \ldots, g\}$, provides the partitioning by assigning a data item $x$ to the group $G_{h(x)}$. The data item $x$ is then said to belong to the group $G_{h(x)}$, and the group's index $h(x)$ is called the *hash value* of the data item $x$. Clearly, there is no unique choice for the hash function $h(\cdot)$. This particular form of hashing is known as *open hashing/closed addressing*, and is one of the two classic forms of hashing [1].

The importance of the hashing technique comes from the fact that, by appropriately selecting the hash function, dictionary operations such as INSERT, DELETE and MEMBER (see [1]) can be carried out in $\mathcal{O}(1)$ time on the average.

In *geometric hashing*, the collection of models to be stored in the database is used during a preprocessing phase (thus executed "off-line") in order to build the hash table data structure. The hash function is selected in such a way that the resulting hash table structure encodes *geometric* information pertaining to small subsets of model features. This geometric information is encoded in a highly-redundant way. During the recognition phase, when presented with a scene from which features are extracted, the hash table data structure is used to

index geometric properties of the scene features to candidate matching models. A search over the scene features is still required. However, the geometric hashing scheme obviates a search over the models and the model features as is the case with the interpretation tree and alignment methods. In the following section, we examine in more detail the geometric hashing idea.

### 2.2.1 Geometric Hashing for Model Matching

Features such as points, linear and curvilinear segments, corners, etc. are extracted during the feature extraction stage (see section 1.1.2). Any such collection of features can be represented by a set of dots: each dot represents the feature's location; associated with each dot is a list of one or more attributes (the feature's *attribute list*) which depends on the corresponding feature's type.

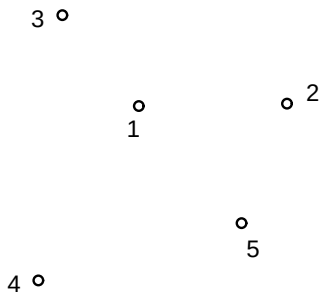


**Figure 2.1 Model $M_1$ consisting of five points.**

We can thus confine ourselves, without any loss of generality, to the problem of recognition of clusters of *point* features (dot patterns), with the understanding that each such point may have associated with it an attribute list. In the simplest case which is examined below, one is interested only in the positional information of the features, and the attribute list is empty. In a more general vision system, one wishes to recognize patterns of lines, corners, and other features, attached to

$3D$ objects undergoing rigid $3D$ transformations and perspectively projected onto the image plane. The geometric hashing algorithms extend to that case as well, at the expense of more complicated transformation classes and implementation issues.

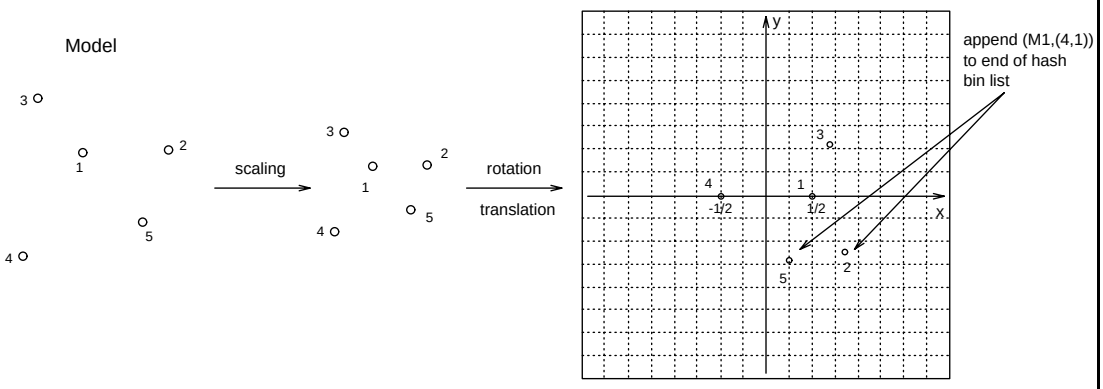


**Figure 2.2 Determining the hash table entries when points 4 and 1 are used to define a basis. The models are allowed to undergo rotation, translation and scaling.**

Suppose that we wish to perform recognition of patterns of point features that may be translated, rotated and scaled (similarity transformations). Two points are needed to define a basis. Figure 2.1 shows a model ($M_1$) consisting of five dots with position vectors $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4$ and $\mathbf{p}_5$ respectively. We begin by scaling the model $M_1$ so that the magnitude of $\overrightarrow{\mathbf{p}_4\mathbf{p}_1}$ in the $Oxy$ system is equal to 1. Suppose now that we place the midpoint between dots "4" and "1" at the origin of a coordinate system $Oxy$ in such a way that the vector $\overrightarrow{\mathbf{p}_4\mathbf{p}_1}$ has the direction of the positive $x$-axis. The remaining three points of $M_1$ will land in three locations. Let us record in a *quantized* hash table, in each of the three bins where the remaining points land, the fact that model $M_1$ with basis "$(4, 1)$" yields

an entry in this bin. This is shown graphically in Figure 2.2.

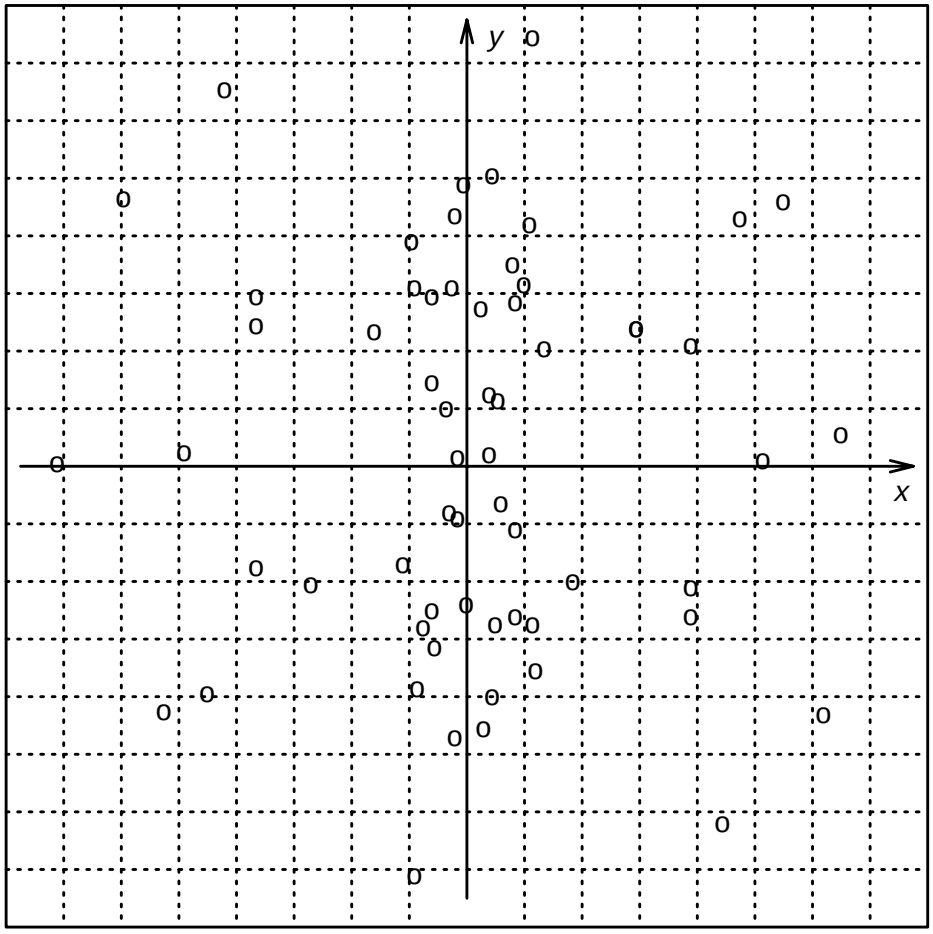


**Figure 2.3 The locations of the hash table entries for model $M_1$. Each entry is labeled with the information "model $M_1$" and the basis pair (i, j) that was used to generate the entry. The models are allowed to undergo rotation, translation and scaling.**

Similarly, the hash table contains three entries of the form $(M_1, (4, 2))$, three entries of the form $(M_1, (4, 3))$, etc. Each triplet of entries is generated by first scaling the model $M_1$ so that the corresponding basis has unit length in the $Oxy$ coordinate system, and then by placing the midpoint of the basis at the origin of the hash table in such a way that the basis vector has the direction of the

positive $x$-axis. The same process is repeated for each ordered basis, and each of the models in the database. Of course, some hash table bins may receive more than one entry. As a result, the final hash table data structure will contain a list of entries of the form $(model, basis)$ in each hash table bin. Figure 2.3 shows the locations of all the hash table entries for model $M_1$.

In the recognition phase, a pair of points, $(\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_2})$, from the image is chosen as a candidate basis. This ordered basis defines a coordinate system $Oxy$ whose center coincides with the midpoint of the pair; the direction of the basis vector $\mathbf{p}_{\mu_2} - \mathbf{p}_{\mu_1}$ coincides with that of the positive $x$-axis. The magnitude of the basis vector defines the "unit" length for $Oxy$.

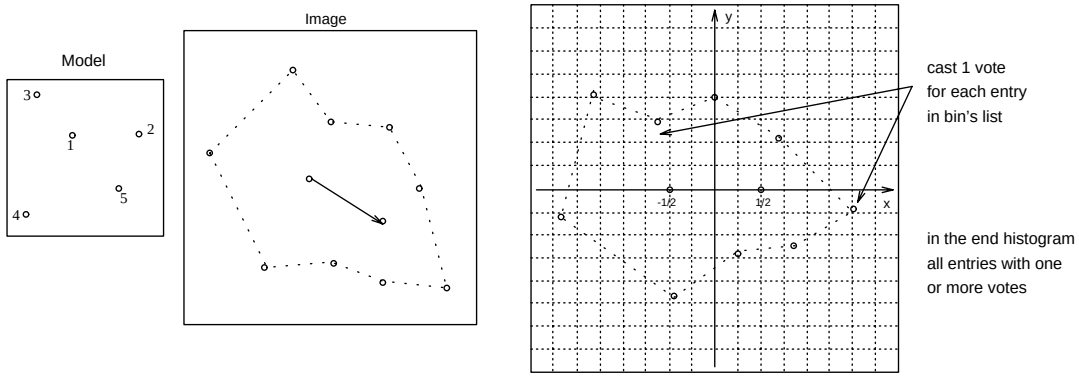


**Figure 2.4 Determining the hash table bins that are to be notified when two arbitrary image points are selected as a basis. The allowed transformation is similarity.**

The coordinates of all other points are then calculated in the coordinate system defined by the chosen basis. Each of the remaining image points is mapped to the hash table, and all entries in the corresponding hash table bin receive a vote. Figure 2.4 shows this graphically. If there are sufficient votes for one or

more $(model, basis)$ combinations, then a subsequent stage attempts to verify the presence of a model with the designated basis matching the chosen basis point. In the case where model points are missing from the image because they are obscured, recognition is still possible, as long as there is a sufficient number of points hashing to the correct hash table bins. The list of entries in each hash table bin may be large, but because there are many possible models and basis sets, the likelihood that a single model and single basis set will receive multiple votes is quite small, unless a configuration of transformed points coincides with a model. In general, we do not expect the voting scheme to give only one candidate solution (see [60]). The goal of the voting scheme is to act as a sieve and reduce significantly the number of candidate hypotheses for the verification step.

*For the algorithm to be successful it is sufficient to select as a basis tuple any set of image points belonging to some model.* It is not necessary to hypothesize a correspondence between specific model points, and specific scene points, since all models and basis pairs are redundantly stored within the hash table. Classification or perceptual grouping of features can be used to make the search over scene features more efficient, for example, by making use of only special basis tuples.

#### 2.2.1.1 The Steps Behind the Idea

We described above how one conceptually determines the index of a hash table bin during the two phases of geometric hashing for the special case of similarity transformations. We next examine in greater detail the actual algorithm, and also give the hash functions for the set of transformations that we will be considering in this dissertation. Note that since both the model and the input data come from a $2D$ grayscale image, the patterns of point features will be planar.

**Case of Rigid Transformations.** Assume that the patterns of point features corresponding to the different models can undergo only rigid transformations, i.e., rotation and translation. A rigid transformation of any such pattern can be uniquely defined by the transformation of two points.

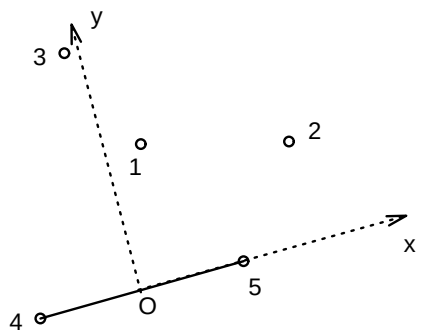


**Figure 2.5 The coordinate system defined by a two-point basis.**

Assume that we are given a set of $n$ point features belonging to one of the models of our database, e.g. $M_1$, and let $\mathbf{p}_{\mu_1}$ and $\mathbf{p}_{\mu_2}$ be an ordered pair of points from that set. Then the vectors $\mathbf{p}_x^r \triangleq (\mathbf{p}_{\mu_2} - \mathbf{p}_{\mu_1}) / \parallel (\mathbf{p}_{\mu_2} - \mathbf{p}_{\mu_1}) \parallel$ and $\mathbf{p}_y^r \triangleq \mathrm{Rot}_{90}(\mathbf{p}_x^r)$[1] form an orthonormal basis, and thus a coordinate system $Oxy$ (see Figure 2.5). Any point $\mathbf{p}$ in the plane can be represented in this basis, namely, there is a unique pair of scalars $(u, v)$, such that

$$\mathbf{p} - \mathbf{p}_0^r = u\mathbf{p}_x^r + v\mathbf{p}_y^r \tag{2.1}$$

where $\mathbf{p}_0^r = \frac{\mathbf{p}_{\mu_1} + \mathbf{p}_{\mu_2}}{2}$ is the midpoint between $\mathbf{p}_{\mu_1}$ and $\mathbf{p}_{\mu_2}$. In other words, the center of the coordinate system $Oxy$ coincides with $\mathbf{p}_0^r$.

The parameters $u$ and $v$ are the coordinates of the point $\mathbf{p}$ in the coordinate system defined by the basis pair. If we apply a rigid transformation $\mathbf{T}$ to all of

[1] I.e. $\mathbf{p}_y^r$ is the vector obtained by rotating $\mathbf{p}_x^r$ counter-clockwise by 90 degrees.

the $n$ points, then with respect to the new basis $(\mathbf{Tp}_x^r, \mathbf{Tp}_y^r)$, the *transformed* coordinates of $\mathbf{Tp}$ will again be $(u, v)$. That is, $(u, v)$ is invariant with respect to transformation $\mathbf{T}$, providing the corresponding points are chosen as the basis pair, and providing the corresponding point is represented in that basis.

We will represent the object points by their coordinates in *all* possible basis tuples. This is because for any given tuple, it is possible that one of the points will be obscured, and thus the specified tuple may not be present. More specifically, the following preprocessing is carried out:

**Preprocessing Phase**

For each model $m$ do:

(1) Extract the model's point features. Assume that $n$ such features are found.

(2) For each ordered pair $(\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_2})$ of point features do:

(a) Compute the coordinates $(u, v)$ of the remaining features in the coordinate frame defined by the basis $(\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_2})$.

(b) After a proper quantization, use the tuple $(u, v)$ as an index to a two-dimensional hash table data structure, and insert in the corresponding hash table bin the information $(m, (\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_2}))$, namely, the model number and the basis tuple which was used to determine $(u, v)$.

If $M$ models are to be inserted in the database, it is clear that the time complexity of this step is $\mathcal{O}(Mn^3)$.

During the recognition phase, the algorithm uses the hash table data structure that was prepared during the preprocessing phase. This phase proceeds as follows:

**Recognition Phase**

When presented with an input image,

(1) Extract the various points of interest. Assume that $\mathcal{S}$ is the set of the interest points found; let $S$ be the cardinality of $\mathcal{S}$.

(2) Choose an arbitrary ordered pair, $(\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_2})$, of interest points in the image.

(3) Compute the coordinates of the remaining interest points in the coordinate system $Oxy$ that the selected basis defines.

(4) For each such coordinate check the appropriate hash table bin, and for every entry found there, cast a vote for the model and the basis.

(5) Histogram *all* the hash table entries that received one or more votes during step (4). Proceed to determine those entries that received more than a certain number (threshold) of votes: each such entry corresponds to a potential match.

(6) For each potential match discovered in step (5), consider all the model-image feature pairs which voted for the particular entry, and recover the transformation $\mathbf{T}$ that results in the best *least-squares* match between all these corresponding feature pairs. Since the computation of this transformation is based on more than two point feature pairs, it will most likely be more accurate.

(8) Transform the edges and higher-order features of the model according to the recovered transformation $\mathbf{T}$ and **verify** them against the input image features. If the verification fails, go back to step (2) and repeat the procedure using a different image basis pair.

The time complexity for the recognition phase is $\mathcal{O}(S^2Mn^3)$, in the worst case. This complexity results from the fact that all $\mathcal{O}(S^2)$ possible image basis pairs may have to be considered, requiring the histograming of $\mathcal{O}(Mn^3)$ data items each time. In reality, though, a handful of basis selections will suffice, and only a small percentage of all hash table entries will be histogramed for each such selection.

**Case of Similarity Transformations.** Assume now that the patterns of point features corresponding to the different models can undergo similarity transformations, i.e. rotation, translation, and scaling. Again, a similarity transformation of any such pattern can be uniquely defined by the transformation of two points.

The analysis is the same as in the case of the rigid transformation, with one simple modification: in the calculation of the transformed coordinates of a point with respect to a basis pair, we normalize the basis pair to have unit length. The orthonormal basis of the system $Oxy$ (see Figure 2.5) defined by the basis pair will now be $\mathbf{p}_x^s \triangleq (\mathbf{p}_{\mu_2} - \mathbf{p}_{\mu_1})$ and $\mathbf{p}_y^s \triangleq \mathrm{Rot}(\mathbf{p}_y^s)$ respectively. Any point $\mathbf{p}$ in the plane can be represented in this basis, namely, there is a unique pair of scalars $(u, v)$ such that

$$\mathbf{p} - \mathbf{p}_0^s = u\mathbf{p}_x^s + v\mathbf{p}_y^s \tag{2.2}$$

where $\mathbf{p}_0^s = \frac{\mathbf{p}_{\mu_1} + \mathbf{p}_{\mu_2}}{2}$ is the midpoint between $\mathbf{p}_{\mu_1}$ and $\mathbf{p}_{\mu_2}$. In other words, the center of the coordinate system $Oxy$ coincides with $\mathbf{p}_0^s$.

The operations during the preprocessing and recognition phases are precisely the same as described in the case of rigid transformations, with the understanding that Eqn. 2.2 is now used to determine the invariants $(u, v)$. The time complexities of the preprocessing and recognition phases remain the same as in the case of the

rigid transformation.

**Case of Affine Transformations.** The case where the patterns of point features corresponding to the different models can undergo a general linear (affine) transformation is slightly different. An affine transformation of any such pattern can be uniquely defined by the transformation of three, instead of two, points.

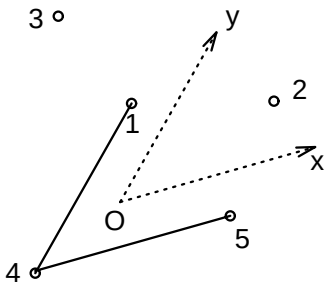


**Figure 2.6 The coordinate system defined by a three-point basis.**

Assume that we are given a set of $n$ point features belonging to one of the models of our database, and let $\mathbf{p}_{\mu_1}$, $\mathbf{p}_{\mu_2}$, and $\mathbf{p}_{\mu_3}$ be an ordered triplet of points from that set. Then the vectors $\mathbf{p}_x^a \triangleq (\mathbf{p}_{\mu_2} - \mathbf{p}_{\mu_1})$ and $\mathbf{p}_y^a \triangleq (\mathbf{p}_{\mu_3} - \mathbf{p}_{\mu_1})$ form a skewed basis, and thus a skewed coordinate system $Oxy$ (see Figure 2.6). Any point $\mathbf{p}$ in the plane can be represented in this basis, namely, there is a unique pair of scalars $(u, v)$, such that

$$\mathbf{p} - \mathbf{p}_0^a = u\mathbf{p}_x^a + v\mathbf{p}_y^a \tag{2.3}$$

where $\mathbf{p}_0^a = \frac{\mathbf{p}_{\mu_1}+\mathbf{p}_{\mu_2}+\mathbf{p}_{\mu_3}}{3}$ is the *barycenter* of the triangle formed by $\mathbf{p}_{\mu_1}$, $\mathbf{p}_{\mu_2}$ and $\mathbf{p}_{\mu_3}$. In other words, the center of the coordinate system $Oxy$ coincides with $\mathbf{p}_0^a$.

The parameters $u$ and $v$ are the coordinates of the point $\mathbf{p}$ in the coordinate system defined by the skewed basis. If we apply an affine transformation $\mathbf{T}$ to all

of the $n$ points, then with respect to the new basis $(\mathbf{Tp}_x^a, \mathbf{Tp}_y^a)$, the *transformed* coordinates of $\mathbf{Tp}$ will again be $(u, v)$. That is, $(u, v)$ is invariant with respect to the transformation $\mathbf{T}$, providing the corresponding triplet of points is chosen to define the skewed coordinate system, and providing the corresponding point is represented in that basis.

The operations during the preprocessing and recognition phases are precisely the same as described in the case of rigid transformations, with the understanding that: (i) Eqn. 2.3 is now used to determine the invariants $(u, v)$, and (ii) the basis tuple is a triplet. The time complexity of the preprocessing phase for the case of the affine transformation is now $\mathcal{O}(Mn^4)$, whereas that of the recognition phase is $\mathcal{O}(S^3Mn^4)$.

**Some General Observations.** Before we conclude this section, we make several observations. The first concerns the selection of the position of the center of the coordinate system $Oxy$ which is defined by the chosen basis. In all three cases above, we made a seemingly arbitrary selection for that position. As we will see in section 4.3, there is a well-founded justification behind our choice.

| Name | $c$ | Allowed Transformation | Preproc. Compl. | Recog. Compl. |
|---|---|---|---|---|
| Fixed | 1 | Translation | $\mathcal{O}(Mn^2)$ | $\mathcal{O}(S^2Mn^2)$ |
| Rigid | 2 | Rotation/Translation | $\mathcal{O}(Mn^3)$ | $\mathcal{O}(S^3Mn^3)$ |
| Similarity | 2 | Rotation/Translation/Scaling | $\mathcal{O}(Mn^3)$ | $\mathcal{O}(S^3Mn^3)$ |
| Affine | 3 | Linear/Similarity | $\mathcal{O}(Mn^4)$ | $\mathcal{O}(S^4Mn^4)$ |

**Table 2.1 The time complexities for the preprocessing and recognition phases, for several transformations.**

The second observation concerns the time complexities of the preprocessing and recognition phases. If $c$ denotes the cardinality of the basis tuple, then the

time complexity of the preprocessing phase is $\mathcal{O}(Mn^{c+1})$, whereas the complexity of the recognition phase is $\mathcal{O}(S^{c+1}Mn^{c+1})$. Table 2.1 summarizes these results.

Also, recall from the previous section that in the context of geometric hashing, the hash functions assume values from the set $\mathbb{R}^2$. In general, the hash function in the context of geometric hashing can have the general form

$$\mathbf{h}: \ \mathbb{R}^k \ \rightarrow \ \mathbb{R}^l,$$

with $k, l \in \mathbb{Z}^+$. The hash function $\mathbf{h}$ is then said to be $l$-dimensional, and the *range* of $\mathbf{h}$ will be referred to as the "hash space," or "space of invariants."

## 2.3 Geometric Hashing Systems

We conclude this chapter with a historical summary of the development of geometric hashing methods for object recognition. This discussion is mostly limited to treatments that led directly to the geometric hashing method or used the geometric hashing terminology. It is clear that related ideas and essentially equivalent concepts occurred frequently in the development of object recognition systems. For example, the "feature sphere" used in Chen and Kak's $3D$-POLY [23] is essentially a hash function that permits indexing into a smaller set of models. Likewise, work by researchers at IBM T.J. Watson Research Center has been based on ideas of indexing into model bases for many years [14,20,21].

Geometric hashing shares certain philosophical underpinings with Hough transform methods, and was in part motivated by related works [5,6]. The difference is that Hough transform methods search over the space of transformations, whereas hashing uses model/scene matches to establish interpretations. The idea of geometric hashing, at least in its modern incarnation, has its origins in work of

Professor Jacob Schwartz [52]. The first efforts were concentrated on the recognition of $2D$ objects from their silhouettes. Hence, efficient curve-matching techniques were developed. The use of "footprints" to describe properties along the curves was later extended by Wolfson and Hong [42] and resulted in a recognition system that was able to recognize about ten $2D$ objects partially occluding each other. The objects were taken from a library of a hundred models, and recognition was performed allowing planar rigid motion (rotation and translation). A landmark in the application of curve matching and combinatorial optimization methods was their use to assemble (graphically rather than physically) all the pieces of two hundred-piece commercial jigsaw puzzles, from separate photographs of their individual pieces [101]. The assembly was based on shape information only. In all two-dimensional curve-matching work, footprints were used to limit the number of candidate curves accessed by the matching system.

However, hash functions (still called *footprint information*) were much more essential when used for $3D$ curve matching obtained from depth data of objects. Using depth data obtained from a fast but approximate depth sensor [22], Wolfson and Kishon developed a practical method for locating and matching curves on rigid $3D$ objects [54], and extended the work by using a different hashing technique [84]; all $3D$ curve-matching systems used measures of the local curvature as index values into a table.

Application of the geometric hashing idea as an approach to model-based vision object recognition was introduced by Lamdan, Schwartz, and Wolfson. Much of the work is summarized in the dissertation of Lamdan [58]. Efficient algorithms were developed for recognition of flat rigid objects assuming the affine approximation of the perspective transformation [61,62] and the technique was

also extended to the recognition of arbitrary rigid $3D$ objects from single $2D$ images [59]. An integrated discussion of both the object-matching and the curve-matching aspects can be found in [45], and related demonstrations of geometric hashing applications are described in [61,62].

Geometric hashing systems have since been built and explored by many research groups. It is fair to say that most implementations of geometric hashing systems seem to work as well as classical model-based vision systems, and deliver in terms of the promise of greater efficiency.

Stein and Medioni [89] present a system for the recognition of planar objects from intensity images. The system uses a hash table that contained the gray-encodings of groups of consecutive edge segments ("supersegments"), of varying cardinalities. For recognition of general $3D$ objects from single $2D$ images, promising results have been obtained in the dissertation of Lamdan [58], where many viewpoint-centered models are generated of simple $3D$ models, and the work of Gavrila and Groen [34], who generate viewpoint-centered models based on experiments that determine limits of discriminability. Stein and Medioni's TOSS system [90] uses "structural hashing" to recognize $3D$ shapes from dense range data from which characteristic curves and local differential patches are extracted. The method allows only for rigid transformations (rotation and translation), and despite the use of high-dimensional indices, the verification stage is very costly.

Forsyth *et al.* [33] present and use descriptors based on *pairs* of planar curves; the descriptors are invariant under affine and perspective transformations. They obtain good results, but their method is sensitive to occlusion and its performance depends strongly on the quality of the segmentation.

For the recognition of rigid $3D$ curves extracted from medical imagery, Guéziec

makes use of hashing methods to speed matching based on spline curve approximations [40]. For recognition of $3D$ objects from range data, Flynn and Jain [32] use hashing and local feature sets to generate hypotheses (without a voting procedure), and report a more efficient search than a constrained search (hypothesize and verify) approach, when applied to two dozen models. Representing $3D$ objects by means of their characteristic view, and regarding object recognition as a graph matching problem, Sossa and Horaud [86] develop hash functions for graphs to provide rapid matching capabilities.

Parallel implementations of geometric hashing, for a Connection Machine, have been attempted by Medioni [17]. A parallel implementation developed for this dissertation has been described elsewhere [78,79,80,81].

There have been numerous efforts to add an error model to geometric hashing, and to investigate its performance in the presence of positional noise of the features. Costa *et al.* [25] investigate the variation of the standard hash functions in the presence of noise, and suggest a weighted-voting scheme. Lamdan and Wolfson [60] investigate both analytically and empirically the false-alarm rate, and conclude that acceptable filtering is possible, although a degradation in performance can be expected for affine-invariant matching. Grimson and Huttenlocher [36] give pessimistic predictions for affine-invariant matching using geometric hashing. Rigoutsos and Hummel [79,80] look at error rates in the presence of noise for both similarity and affine invariance, and conclude that weighted voting is essential for recognition in the affine case. Gavrila and Groen [34] report good filtering capabilities of similarity-invariant model matching in the presence of noise. Fischer *et al.* apply the geometric hashing method to the problem of structural comparison of proteins [29].

There is also a number of corporate research groups that are using geometric hashing methods for US Defense Department-funded projects. We are aware of geometric hashing investigations at I-Math Associates, in Orlando, Florida, at Martin Marietta Denver, and at The Analytic Sciences Corporation.

# Chapter 3

# Exploiting Parallelism

In this chapter, we examine the issue of parallelism in the context of geometric hashing. Two parallel algorithms that realize the geometric hashing method are described. In addition to these two algorithms, a novel hypercube-based *radix-sort* algorithm as well as a number of general performance enhancements are presented.

The first algorithm is based on a "connectionist" view of the geometric hashing method and is designed for an SIMD hypercube-based machine. The algorithm is data parallel over the hash table entries and regards geometric hashing as a connectionist algorithm with information flowing via patterns of communication.

The second algorithm is more general, and relies on a "data broadcast" approach. This algorithm is data parallel over *combinations* of small subsets of model features, and is inspired by the method of inverse indexing for data retrieval [87]. The algorithm treats the parallel architecture as a source of "intelligent memory."

Both algorithms are sequential over the observed image features. This is a fundamental design decision and is motivated by the fact that the geometric

hashing method greatly speeds the search over the database containing the models and the anchor points within the models, while still requiring a search over the set of features in the image. However, once a set of candidate features (a basis tuple) is selected such that the features belong to a model embedded in the scene, recognition of the model will follow.

An important step during the recognition phase is the *histograming* of those hash table entries that received one or more votes during the voting process. One of the approaches to histograming is the use of sorting: a novel and simple hypercube-based *radix-sort* algorithm is described.

## 3.1 Parallelizability of Geometric Hashing

One of the major advantages of the geometric hashing method is that it is inherently parallelizable. Its parallel nature manifests itself both in the preprocessing (off-line) and the recognition (on-line) phase of the algorithm. There are several ways that the algorithm can be parallelized, depending on the mapping of data items to the available processing elements (PE's).

Let us recall the description of the preprocessing phase from section 2.2.1. We assume that the database will contain $M$ models each consisting of $n$ point features, and that the models are allowed to undergo similarity transformations (thus, two points are needed to define a basis tuple).

**Parallelism in the Preprocessing Phase.** During the creation of the hash table data structure, and for a given model and basis selection, we compute the hash invariants for each subset of model features comprising the selected basis and each of the remaining $(n-2)$ model features. Clearly, this computation can

proceed in parallel. Depending on the number of available PE's, the computations corresponding to the $n(n-1)$ distinct ordered bases possible may also proceed in parallel. Each of the $n(n-1)(n-2)$ PE's will compute one hash location in $\mathcal{O}(1)$ time. Finally, if at least $Mn(n-1)(n-2)$ PE's are available, the hash invariants corresponding to all possible $(basis,\ model-feature)$ combinations can be computed in $\mathcal{O}(1)$ parallel time.

Once a set of hash invariants is available, the relevant $(model, basis)$ information can be stored in the appropriate hash bins, again in parallel. This may result in contention, when more than one PE's, having computed the same hash invariant, attempt to deposit distinct $(model, basis)$ tuples in the same hash bin. A simple protocol can be used to impose an arbitrary ordering on the competing PE's.

**Parallelism in the Recognition Phase.** During the recognition phase, $S$ features from the feature extraction process are presented. Once a basis has been selected, hash invariants must be computed for each set of features composed of the selected basis and one of the remaining $S-2$ scene features. Assuming the availability of at least $S$ PE's, the computation of these invariants can proceed in parallel and the appropriate hash bins are notified. If there is at least as many PE's as hash bins, then the list of entries associated with each bin can be stored in the local memory of a PE which is assumed to control that bin. The local lists can then be traversed in parallel, with the longest such list dominating the required time for the list traversal. Finally, as we will see in section 3.4, the histograming of the votes received by the various $(model, basis)$ combinations can also proceed in parallel.

In the above description, we use only one basis pair during the recognition phase. However, by repeating the coordinates of the $S$ interest points, we can employ multiple bases at once, at the expense of additional bookkeeping on a per processor basis. For example, if the input images have a maximum of 200 interest points, as many as 256 bases can be computed at the same time on a $64K$-processor machine. The PE assigned to a hash bin must maintain a separate counter for each of the 256 bases. The $i$-th such counter counts the number of times that the bin is hit by invariants that are computed in the coordinate system defined by the $i$-th basis. During the histograming step, messages must be "tagged" with one of 256 possible identifiers, before performing the evidence accumulation.

## 3.2 Some Definitions

We will now give several definitions that will facilitate the description of the two algorithms.

Typically, two separate components can be (logically and/or physically) identified in any SIMD supercomputer. One component is the array of processors executing the various instructions. The other component is a traditional sequential computer that *controls* the array of processors. The role of the sequential computer is to provide a familiar environment to the users of the supercomputer for program development and execution. User programs execute on the sequential computer which translates the segments of code that are to be executed in parallel into sequences of instructions plus data; these sequences are subsequently relayed to the various processors of the array with the help of a communication network. We will be referring to the sequential machine as the *host*.

SIMD supercomputers are the typical target architectures for implementing data parallel algorithms. Data parallelism separates tasks to be performed concurrently according to the indices of items in the data structures participating in the algorithm. This indexing effectively associates one data item with one processor from the processor array. The number of processors comprising the array is typically limited to a few tens of thousands, whereas in many applications the data sets have cardinalities that are in the order of millions of items. For this reason, software support provides the user with a *virtual processor* facility. This facility multiplexes the physical processors of the processor array in order to *simulate* a machine with many more than the actually available processors. The local memory of each physical processor is divided evenly among the *virtual* processors it simulates, and all physical processors simulate the same number of *virtual* processors. The number of virtual processors simulated by one physical processor will be referred to as the "*virtual processor ratio*," or "VPR" [95]; a VPR value of $r$ incurs an *at least* $r$-fold slowdown in execution speed.

Finally, many applications consist of more than one data set that are mapped to the array of processors during the lifetime of the program. We will call the set of virtual processors to which a given data set is mapped a "*virtual processor set*," or "VP set" [95] for short. At any time during program execution, more than one VP set may be in existence; these VP sets will correspond to distinct data sets. Clearly, the physical processors simulating the different VP sets will not be disjoint, implying that at most one VP set can be active at any moment. Distinct VP sets can communicate with one another either through message passing, or by sharing variables.

## 3.3 Design Issue

Three data sets that can be identified in the description of the recognition phase (section 3.1) are: the set of the coordinates of the input image features, the set of the coordinate tuples corresponding to all possible basis tuples formed from image features, and the set of hash table entries.

The discussion implicitly assumed the following mapping:

- The coordinates of the image features comprised the first VP set;
- The hash entries belonging to a given hash bin were coalesced together into one group; each such group was mapped to one processor, thus forming the second VP set.

The coordinates of a selected basis are broadcast by the host to the processors that implement the first virtual processor set. This approach makes the algorithm data parallel over the image features, and serial over the set of basis tuples.

Alternatively, each member of the set of basis tuples could be associated with one processor; the coordinates of the image feature points could then either be broadcast by the host, or held in temporary storage. This mapping results in a different approach to performing geometric hashing. This latter approach is data parallel over the set of basis tuples, and serial over the image features.

As can be seen, for a given problem, there is relative flexibility in deciding which data sets are mapped to the available processors, and how. In our description of the parallel algorithms for performing geometric hashing, an alternative mapping will be presented.

## 3.4 Building-block Algorithms

Before we proceed with the description of the algorithms for performing geometric hashing in parallel, we present certain building-block algorithms that are fundamental to the programming of a hypercube-based SIMD architecture and which will be used as "subroutines" by our algorithms. We assume a concurrent-read-exclusive-write (CREW) model of computation: any pattern of concurrent reads to neighboring processors uses unit time. (Accesses are permitted along different dimensions in the same clock cycle).

### 3.4.1 The $p$-product

The first of the building block algorithms is the one needed to perform a $p$-product. The $p$-product is defined as follows: given $p$ finite sets $A_1 = \{a_{1,i_1}\}_{i_1=1}^{\ell_1}$, $A_2 = \{a_{2,i_2}\}_{i_2=1}^{\ell_2}$, $A_3 = \{a_{3,i_3}\}_{i_3=1}^{\ell_3}$, ..., $A_p = \{a_{p,i_p}\}_{i_p=1}^{\ell_p}$, the $p$-product $A_1 \times A_2 \times \ldots A_p$ is the set of all the ordered $p$-tuples $\{(a_{1,i_1}, a_{2,i_2}, \ldots, a_{p,i_p})\}_{i_1=1,i_2=1,\ldots,i_p=1}^{\ell_1,\ell_2,\ldots,\ell_p}$.

One way to compute the $p$-product is to perform an outer product $p-1$ times. An outer product for the Connection Machine is succinctly described in [66]. An extension of the method leads to a direct $p$-product computation, which we describe next.

Using standard Gray-code embedding algorithms, we configure the hypercube as a $p$-dimensional array of size $\ell_1$ by $\ell_2$ by ... by $\ell_p$. Let $x_j$, $j = 1, 2, 3, \ldots, p$ denote the $j$-th axis of this configuration. We assume, purely for convenience, that the $\ell_i$'s are powers of 2, and that a sufficient number of processors is available.

The processors are indexed by their coordinates $(i_1, i_2, \ldots, i_p)$ and, initially, data element $a_{1,i}$ is contained in processor $(i, 0, \ldots, 0)$, $a_{2,i}$ in processor $(0, i, \ldots, 0)$

etc. In other words, the elements of set $A_j$ are spread along the $x_j$-axis, $j = 1, 2, \ldots, p$.

The algorithm has two phases. During the first phase, the $a_{1,i}$ data is spread along a row in the direction of the $x_2$-axis, the $a_{2,i}$ data is spread in the direction of the $x_3$-axis etc. In general, the $a_{j,i}$ data is spread along a row in the direction of the $x_{j \bmod k+1}$-axis. Figure 3.1 shows a simple example with three sets $A_1$, $A_2$, and $A_3$ (for the case of a *triple* product).

In the second phase, the data on each hyperplane is spread into the entire hypercube, first spreading the data on the $(x_1, x_2, \ldots, x_{k-1})$-plane along the $x_k$-axis, then the data on the $(x_2, x_3, \ldots, x_k)$-plane along the $x_1$-axis, etc. Finally, the data on the $(x_k, x_1, \ldots, x_{k-2})$-plane is spread along the $x_{k-1}$-axis.

Upon completion, processor $(i_1, i_2, \ldots, i_k)$ will have received datum $a_{1,i_1}$ from $(i_1, 0, \ldots, 0)$, datum $a_{2,i_2}$ from processor $(0, i_2, \ldots, 0)$, etc., and thus has the $p$-product element $(a_{1,i_1}, a_{2,i_2}, \ldots, a_{k,i_k})$.

The operation of spreading data along a single axis that occurs during both phases can clearly be performed in $\mathcal{O}(\ell_j)$ time, since nearest neighbors are adjacent in the hypercube, but can in fact be completed in $\mathcal{O}(\log \ell_j)$ time. This is because we may use a recursive doubling scheme to spread the data rapidly along the axis. (Algorithms of this kind are described by Hillis [41].) In the parlance of the Connection Machine *Paris* (TM) language, the operation is a "scan_with_copy." Power-of-two communication along each axis is provided by $\mathcal{O}(1)$ communication cycles due to the Gray-code embedding. Specifically, if $\mathbb{G}(i)$, $i = 0, 1, \ldots, n-1$, is a Gray-code ($n$ a power of 2), then it can be shown that $\mathbb{G}(i)$ and $\mathbb{G}((i+2^d) \bmod n)$ differ in at most two bits, and thus can be connected by two communications cycles on a hypercube. This is true for any value of $d$.

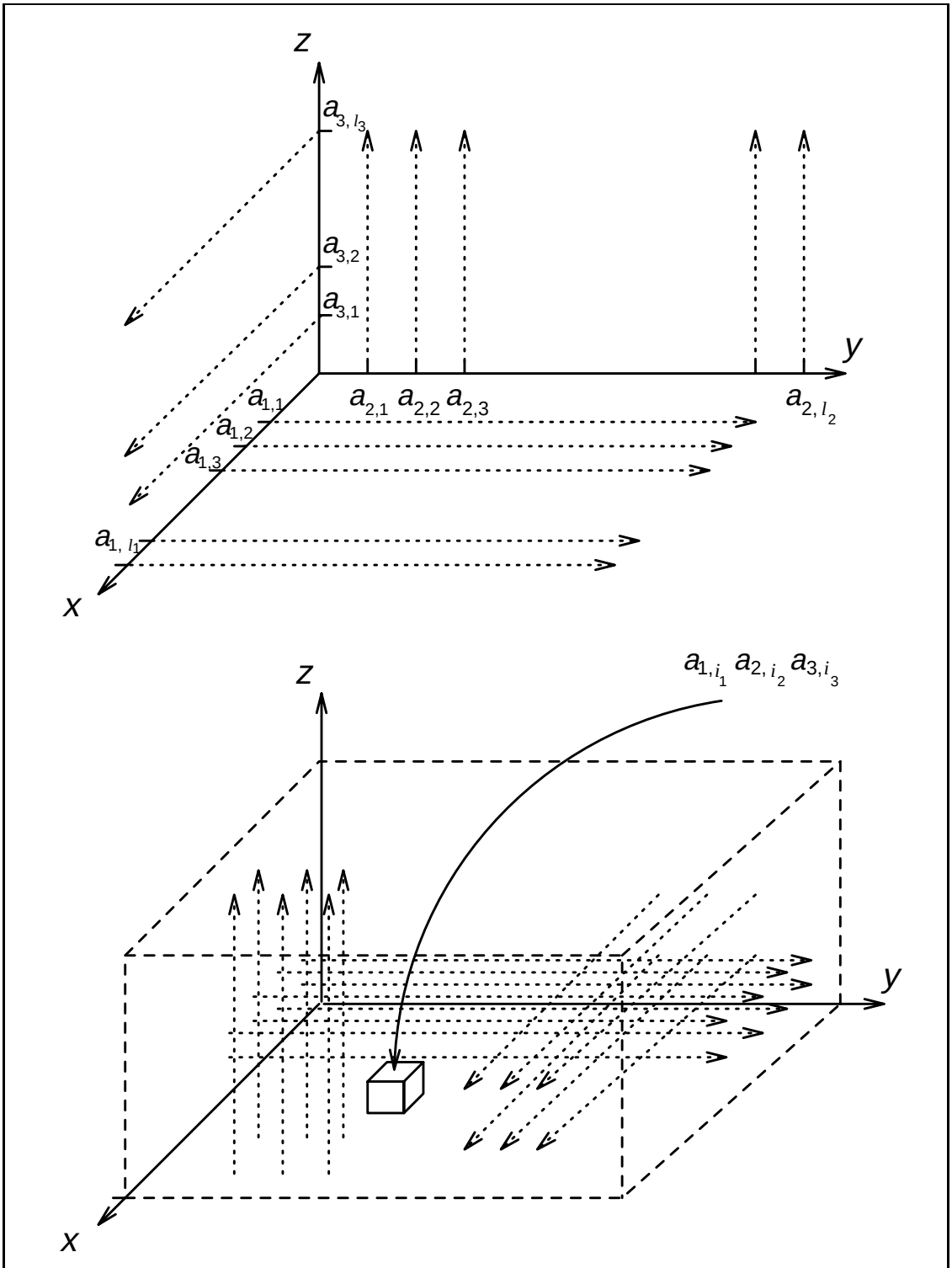


**Figure 3.1 The two stages of the parallel p-product computation for the simple case where p=3 (3-product).**

### 3.4.2 Histograming

The second building-block algorithm is needed to perform histograming in parallel. Histograming can be defined as follows: given a collection of data $\{a_i\}_{i=1}^{D}$, such that each $a_i$ is an element of a finite collection of possible values, say $a_i \in \{1, 2, ..., V\}$, determine a count of the number of elements equal to each possible output value, i.e., $H(k) = \#\{i \mid a_i = k\}$. The value $H(k)$ is also known as the *frequency* of value $k$.

As pointed out in [66], there are three distinct approaches to parallel histograming:

(1) Sequentially iterate, from 1 through $V$; for each value $k$, allow each $a_i$ to mark itself if it is equal to $k$, and then perform a parallel sum to count the number of elements that were marked. Since parallel summing is $\mathcal{O}(\log D)$, this method has parallel complexity $\mathcal{O}(V \log D)$.

(2) Perform additive writes; each processor looks at its value $a_i$, and sends a message to processor $a_i$ to increment an accumulator. There are two virtual processor sets, the initial set with $D$ processors, one per data element $a_i$, and a set of $V$ processors in the output, containing one accumulator per processor. The parallel complexity on an SIMD hypercube without additive writes is $\mathcal{O}(\log^2 D + \log V)$, although additive writes will typically be handled in an average-case more efficient method. In practice, the messages to increment accumulators will be combined in a probabilistic routing network, to avoid serialization at the location of the accumulators. The complexity is in all cases at least $\Omega(\log D)$, since if all messages are destined to a single processor, then the combination of the messages is equivalent to

a global sum, but in practice, the complexity will depend upon $V$ and $D$, the efficiency of the routing algorithm and the combining of messages in the router.

(3) Sort the data, so that $a_{\pi(i)}$ (where $\pi(\cdot)$ is a permutation) forms a nondecreasing sequence, for $i = 1, ...D$. For example, the Batcher bitonic sort algorithm operates on a hypercube machine in $\mathcal{O}(\log^2 D)$ time. After sorting, each processor can determine if the data in the processor to its left is different. If so, it marks itself as the head of a constant-data block. Since each processor needs to be able to communicate with its neighboring processor for this step, the processors should be configured as a one-dimensional array embedded in the hypercube, using a Gray-code embedding. The Batcher sort process is still efficient in this configuration, although with a penalty in the proportionality constant. Next, a segmented parallel prefix sum is used to count the number of processors in each constant-data block and this information is delivered to the head processor of each block. Finally, each head processor sends the information about the cardinality of its block to the appropriate histogram bin, $a_{\pi(i)}$; this is the processor whose index is equal to the data item shared by the processors of the block. Since the destinations of the messages are distinct and ordered relative to the indices of the source indices, these messages can be sent using an $\mathcal{O}(\log D + \log V)$ contention-free algorithm of the sort described by Nassimi and Sahni [73]. The total parallel time complexity of histograming by sorting is thus $\mathcal{O}(\log^2 D + \log V)$.

For our purposes, the histogram vector is not needed; rather, we only need knowledge of the few maximum-vote-getting values. To this end, the final stage

of sending messages (method (3) above) can be omited, and the maximum counts among the marked processors can be determined and relayed to the front end. Thus the process of finding the maximum histogram bin can be accomplished in $\mathcal{O}(\log^2 D)$ time.

In the actual implementation of our system, to be described later, we used method (2) above. However, the most efficient implementation would have been to use method (3) in conjunction with a radix-sort algorithm. Lin and Kumar [65] provide a hypercube-based radix-sort algorithm; however, because they sort from high-order bit to low-order bit, the algorithm is unnecessarily complicated. In the next section, we present a simpler method for performing radix sorting on a hypercube.

#### 3.4.2.1 A Novel Radix-Sort Algorithm

We now describe a novel algorithm for performing radix sorting on a hypercube. This algorithm has the same time complexity as the one described by Lin and Kumar [65], but it is much simpler. The time complexity of the algorithm is $\mathcal{O}(\log V \times \log D)$.

The algorithm is outlined in Figure 3.2, whereas Figure 3.3 illustrates the algorithm for a simple example data set.

## 3.5 The Geometric Hashing Connectionist Algorithm

In this section, we present the first of the two data-parallel algorithms for performing geometric hashing on a hypercube-based SIMD architecture.

The algorithm is based on a "connectionist" view of the geometric hashing

Assume that the values in the sequence $\{a_i\}_{i=1}^{D}$ to be sorted are represented in binary bit form, and let $\{b_{\ell,i}\}_{i=1}^{D}$ be the sequence of the $\ell$-th from-the-right bits. We sort the values in a stable fashion.

For $\ell$ beginning at zero, and successively increasing to $\log V - 1$, we do the following:

a: Mark all processors with $b_{\ell,i} = 0$.

b: Rank these processors: each marked processor determines its relative position among all marked processors using a parallel prefix sum (Nassimi-Sahni [73] describe a RANK algorithm). Let $r_i$ be the rank of a processor if it is marked and $t$ be the maximum $r_i$.

c: Mark all processors with $b_{\ell,i} = 1$.

d: Rank these processors as well; let $s_i$ be the rank of the $i$-th such processor.

e: Move the $a_i$ data: every processor with $b_{\ell,i} = 1$ sends its data $a_i$ to processor $t + s_i$, while every processor with $b_{\ell,i} = 0$ sends its data to processor $r_i$. Because the paths of communication are ordered, this routing can be completed in $\mathcal{O}(\log D)$ time, using the CONCENTRATE and DISTRIBUTE algorithms from [73].

After the first iteration, all items are stably sorted with respect to their low-order bit. Upon termination, the sequence $\{a_i\}_{i=1}^{D}$ will be sorted.

**Figure 3.2 Simple Radix Sort on a Hypercube.**

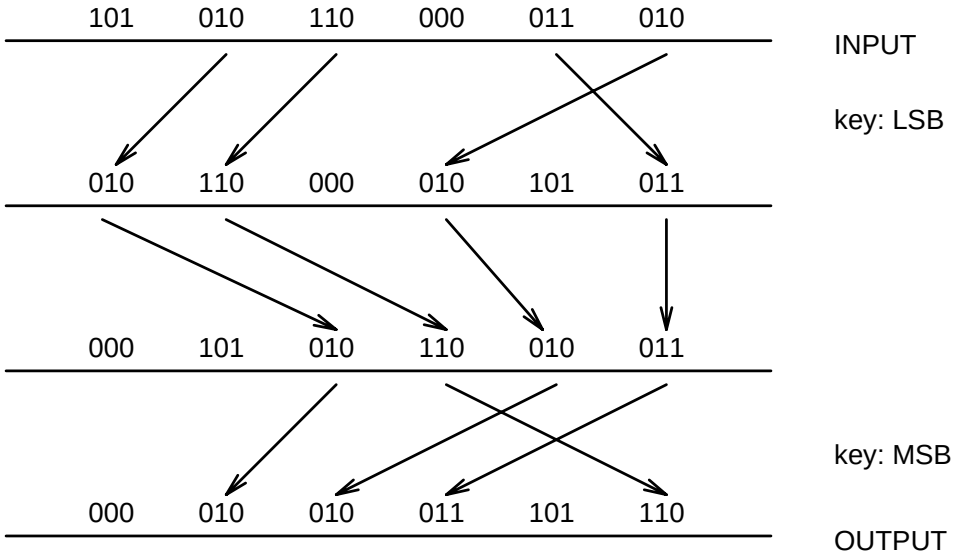


**Figure 3.3 Radix Sort: an illustration.**

method and is data parallel over the hash bin entries and the image features. The database is assumed to contain $M$ models; each model $m$ has an associated set of features, $\mathcal{F}_m = \{\mathbf{f}_{m,k}\}_{k=1}^{n} = \{(x_{m,k}, y_{m,k})\}_{k=1}^{n}$, containing the coordinates of the model's $n$ features. Without loss of generality, the coordinates of the features of all models are assumed to reside in the local memory of the host. The number of features extracted from the input image is $S$, and the hash table consists of $B$ bins. Since we do not wish to restrict ourselves by making any assumptions regarding the transformation that the database models are allowed to undergo, the cardinality of the basis tuple will be a parameter, and equal to $c$. Finally, the availability of $\Theta(Mn^{c+1})$ PE's is assumed.

### 3.5.1 Connectionist Algorithm: Preprocessing Phase

During the preprocessing phase, the hash table is created for the set of $M$ models. Four virtual processor sets participate in this phase: the model feature set $V_1$, the $(c+1)$-product set $V_2$, the hash bin set $V_3$, and the set of hash bin entries $V_4$. This phase is completed in two passes. The purpose of the first pass is to determine the number of entries that will hash to each bin of the hash table, when all the models in the database are considered. During the second pass the hash entries are actually stored in the hash table.

**Preprocessing Phase: Pass 1**

For each model $m$ do:

**Stage 1:** The host relays the coordinates of the $m$-th model's features to the $n$ PE's of set $V_1$: the $i$-th element of set $\mathcal{F}_m$ will reside in the local memory of the $i$-th PE of set $V_1$.

**Stage 2:** The $(c+1)$-product $(\mathcal{F}_m)^{c+1}$ is computed using the $p$-product algorithm described in section 3.4. Each of the $n^{c+1}$ processors of set $V_2$ now contains a $(c+1)$-tuple of the form $\left[(x_{i_1}, y_{i_1}), (x_{i_2}, y_{i_2}), \ldots, \left(x_{i_{(c+1)}}, y_{i_{(c+1)}}\right)\right] \in (\mathcal{F}_m)^{c+1}$. The first $c$ points of each such tuple define an ordered basis and thus a coordinate system. Some of those bases are formed by repeating the same feature point, and thus are degenerate; the corresponding virtual processors will be disabled and will not participate in the rest of the computation. Additionally, those processors in charge of tuples where the $(c+1)$-st point coincides with one of points forming the basis will be disabled as well. The remaining processors proceed to compute the coordinates of the $(c+1)$-st point in the coordinate system defined by the basis. Subsequently, these coordinates are converted to a hash bin number, i.e. the index $h$ of a processor in the two-dimensional set $V_3$.

**Stage 3:** Each processor of $V_2$, with information destined for a certain hash bin, sends an additive write with increment 1 to an accumulator in that bin. This is effected by sending a message to the processor of $V_3$ in charge of the corresponding hash bin. A subsequent parallel prefix operation on the resulting counts allows us to organize the set $V_4$ of hash entries into a one-dimensional array so that entries belonging to the same hash bin occupy a contiguous block of processors. There is a total of $B$ such blocks, and the length of each block is precisely equal to the number of expected entries in the corresponding hash bin. The first processor of a block is assumed to *head* the block. Finally, a map is built: the $h$-th processor of $V_3$ stores locally the index $T_h$ of the $V_4$ processor heading the block of entries for the $h$-th hash bin.

Figure 3.4 details the first pass for the case where the basis tuple consists of two points.

**Preprocessing Phase: Pass 2**

For each model $m$ do:

**Stage 1:** The same as in the first pass.

**Stage 2:** The same as in the first pass.

**Stage 3:** At this point, each of the active processors of $V_2$ has computed the index $h$ of a processor in the two-dimensional set $V_3$; the latter will receive a message requesting the index of the next available processor in the group of $V_4$ processors headed by $T_h$. Clearly, it may happen that more than one processors of $V_2$ send a request to the same $V_3$ processor. A variable in the local memory of the $h$-th processor of $V_3$, initially equal to $T_h$, can provide the necessary information. Combining this with an SIMD version of a parallel *Fetch-and-Add* resolves the contention simply and effectively. This stage concludes with the active processors of $V_2$ sending a tuple of the form $(m, (i_1, i_2, \ldots, i_c))$ to the appropriate processor of $V_4$.

Figure 3.5 details the second pass for the case where the basis tuple consists of two points.

The hash table is represented by two data structures. The first of the two structures (set $V_3$) contains one processor for each hash bin $h$, and gives the pointers to a head processor $T_h$ of a block of data in the second structure (set $V_4$).

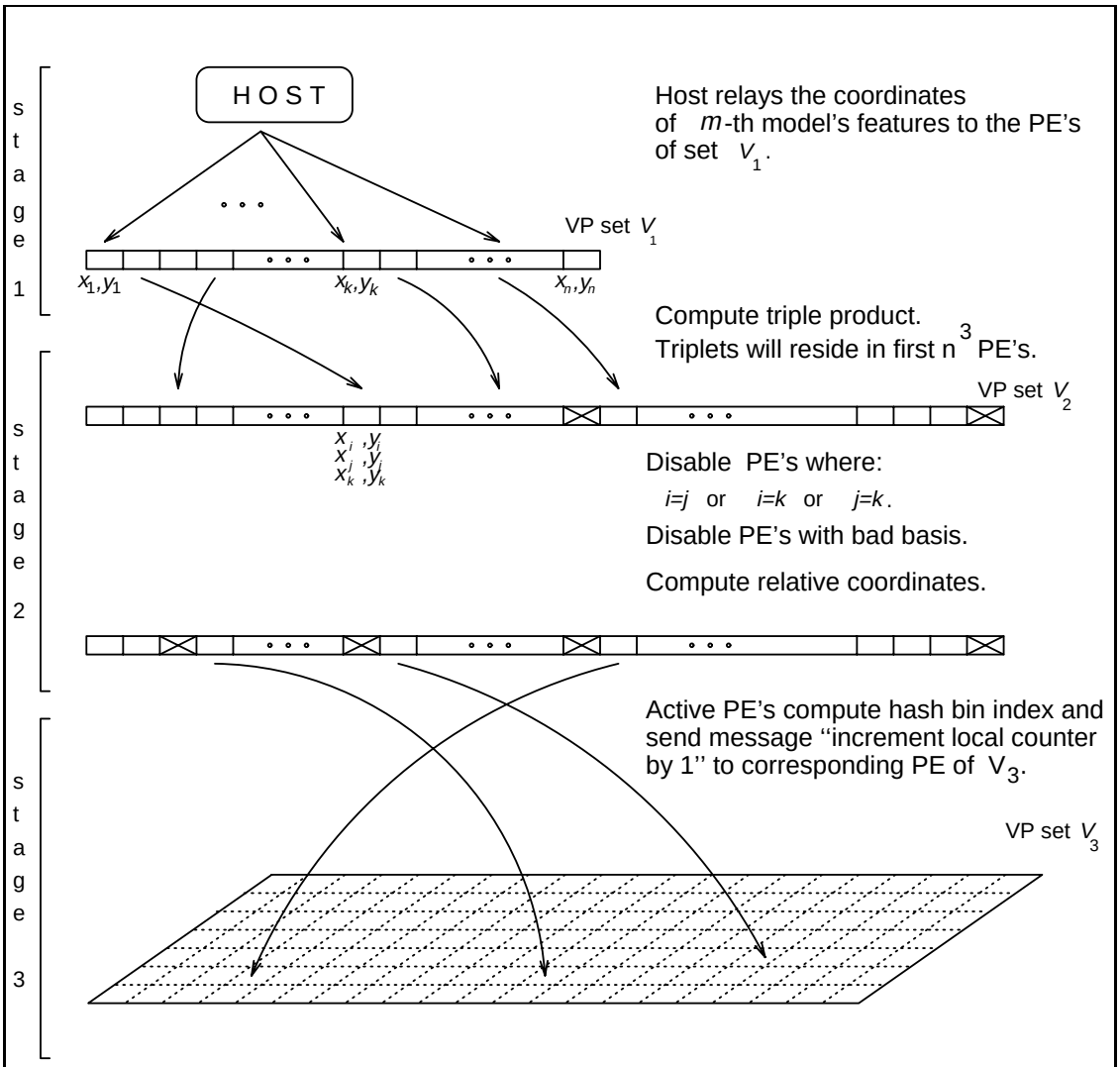


**Figure 3.4** The first pass of the preprocessing phase for the case where the basis tuple consists of two points.

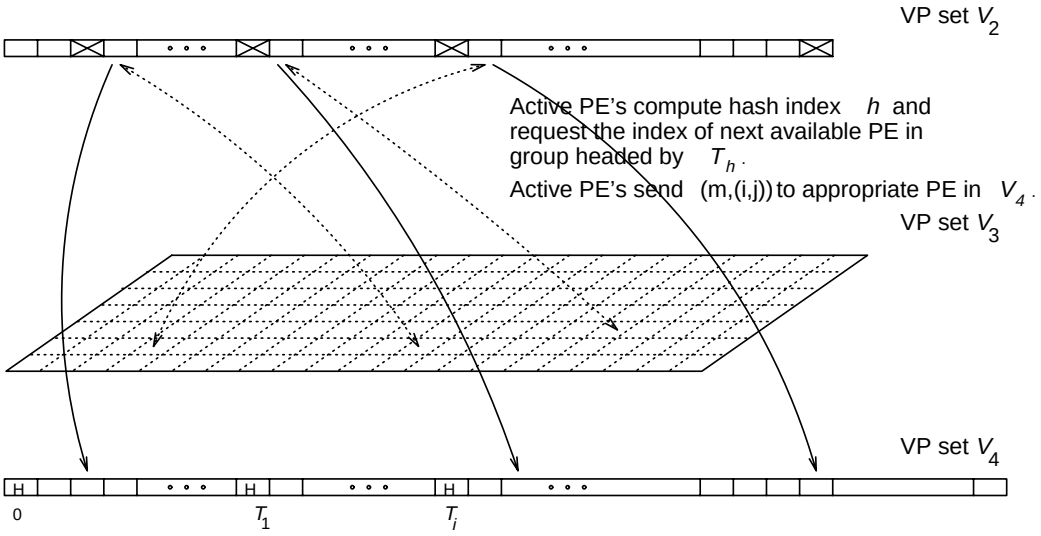


**Figure 3.5 The third stage of the second pass of the preprocessing phase for the case where the basis tuple consists of two points.**

This latter structure consists of at most $M\binom{n}{c}(n-c)\,c!$ hash bin entries which are of the form $(m,(i_1,i_2,\ldots,i_c))$.

### 3.5.2 Connectionist Algorithm: Recognition Phase

Four virtual processor sets participate in the recognition phase: the feature coordinate set $V_1$, the hash table sets $V_2$ and $V_3$, and the histogram bin set $V_4$. We assume that the coordinates of the features that were extracted from the input image reside in the local memory of the $S$ processors of the set $V_1$: the coordinates of the $i$-th image feature reside in the memory of the $i$-th processor of $V_1$. The hash table is preloaded from storage into the local memory of the processors comprising the sets $V_2$ and $V_3$: the VP set $V_2$ contains the pointers to the heads of the blocks of entries whereas $V_3$ is the one-dimensional array of concatenated lists of hash entries. This phase of the algorithm proceeds in three stages.

**Recognition Phase**

**Stage 1:** The host selects a basis tuple (a set of $c$ image features) and relays its coordinates to the $S$ processors of $V_1$. Each processor in $V_1$ combines the coordinates of the feature stored locally with the broadcast tuple to compute the feature's coordinates in the system defined by the basis. The coordinates are subsequently used to determine the index of the hash bin (processor in $V_2$) to be notified.

**Stage 2:** In the second stage, messages saying "you receive one vote" are sent by the processors of $V_1$ to the appropriate processors in $V_2$. The messages are sent using additive writes and the general routing facilities provided by the interconnection network; multiple votes destined for the same recipient processor combine *en route* to the destination.

**Stage 3:** In the last stage, every processor $h$ from the set $V_2$ that received one or more messages in the previous stage relays the number of votes that it received to the block of processors $T_h$ through $T_{h+1} - 1$ of $V_3$. This operation can be done, for example, using a modified version of Nassimi-Sahni's GENERALIZE algorithm [73]. Alternatively, every processor $h$ from set $V_2$ can send a message containing the number of votes (which might be zero) to the processor $T_h$ in $V_3$. Using a parallel prefix computation with "copy from the left" as the binary associative operator, processor $T_h$ can then spread the count to the remaining members of its group. At this point, we wish to histogram the entries of the processors in the set $V_3$ using the multiplicities determined in the previous step. Use of the radix sort algorithm that was described in section 3.4.2.1 offers

advantages from a complexity viewpoint. Each processor of $V_4$ is associated with one histogram bin representing a tuple of the form $(m, (i_1, i_2, \ldots, i_k))$. Upon termination of the histograming step, each of the $M\binom{n}{c}c!$ processors of $V_4$ contains the frequencies of the corresponding $(model, basis)$ combination. A thresholding operation of the vote tallies over the processors of the set $V_4$ recovers the winning $(m, (i_1, i_2, \ldots, i_k))$ combinations. These combinations are communicated back to the host which will verify the existence of matching models.

---

Figure 3.6 details the recognition phase for the case where the basis tuple consists of two points.

### 3.5.3 Time Complexity

For the time complexity of the algorithm, we assume that $M\binom{n}{c}(n-c)c!$ PE's are available. The interconnection network is a hypercube, and the concurrent-read-exclusive-write (CREW) model of computation is adopted. We further assume that $M > n$, $M > S$, and $Mn^{c+1} \gg B$.

**Preprocessing Phase:** The complexity of this phase is dominated by the third stage of the second pass. During that stage, the $\Theta(n^{c+1})$ processors of $V_2$ begin by requesting the index of the processor in $V_4$ that will eventually receive a message of the form $(m, (i_1, i_2, \ldots, i_c))$ to a processor in $V_3$. Although, in principle, it is possible that all the active processors of $V_2$ send their request to the same processor of $V_3$, thus forcing serialization and increasing the time complexity of this step to $\mathcal{O}(n^{c+1} \log B)$, fewer than a small constant number of collisions are

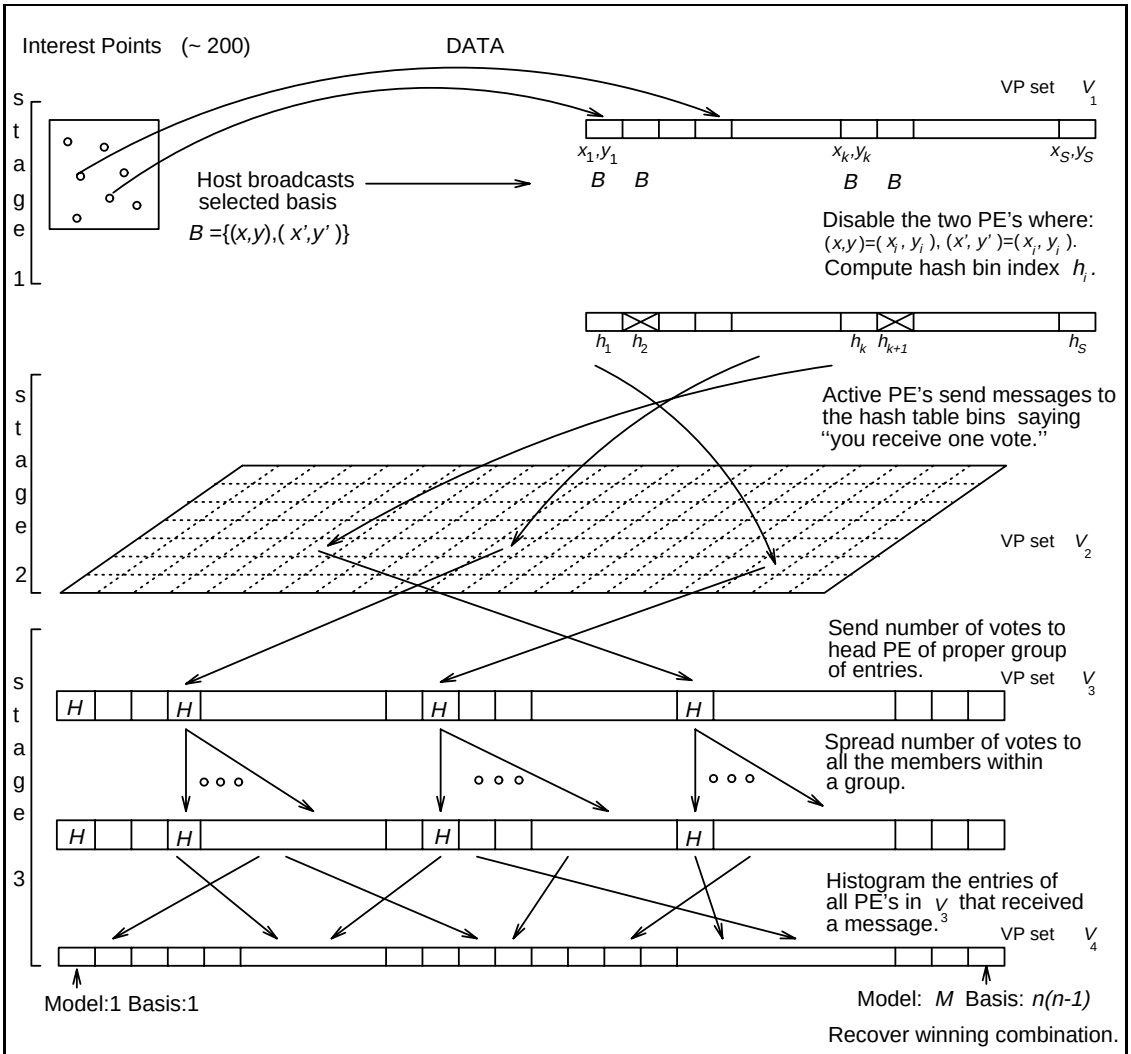


**Figure 3.6** The recognition phase of the parallel geometric hashing connectionist algorithm, for the case where the basis tuple consists of two points. Note how tokens flow from one set via connections to the next set.

expected to occur on the average.[1] As a result, the time complexity of this step is $\mathcal{O}(\log B)$ per database model. The active processors of $V_2$ continue by forwarding a message of the form $(m, (i_1, i_2, \ldots, i_c))$ to the appropriate processor of $V_4$; it should be noted that no two processors of $V_2$ will forward to the same target processor of $V_4$. This second step can be completed in $\mathcal{O}\left(\log\left(Mn^{c+1}\right)\right)$ time per database model. The time complexity of the preprocessing phase is thus $\mathcal{O}\left(M \log\left(Mn^{c+1}\right)\right)$ which is the same as $\mathcal{O}\left(M \log\left(Mn\right)\right)$. The preprocessing phase is very expensive, even when carried out in parallel, but this is of no concern since it is executed off-line.

**Recognition Phase:** The time complexity of the recognition phase, per broadcast basis tuple, is dominated by the histograming step. In fact, the time complexity of the remaining operations of the recognition phase is no worse than $\mathcal{O}\left(\log\left(Mn^{c+1}\right)\right)$ which is the same as $\mathcal{O}\left(\log\left(Mn\right)\right)$. The complexity of histograming depends on the particular method that is used; if Batcher's bitonic sort [76] is used to perform the histograming, the time complexity of the recognition phase is $\mathcal{O}\left(\log^2\left(SMn^{c+1}\right)\right)$ or equivalently $\mathcal{O}\left(\log^2\left(SMn\right)\right)$. On the other hand, if the radix sort algorithm that was described in section 3.4.2.1 is used, the time complexity of the recognition phase drops to $\mathcal{O}\left(\log\left(SMn^{c+1}\right)\log\left(Mn^{c}\right)\right)$ or $\mathcal{O}\left(\log\left(SMn\right)\log\left(Mn\right)\right)$.

## 3.6 The Hash-location Broadcast Algorithm

In this section, we present the second of the two data-parallel algorithms for performing geometric hashing.

[1] This, in fact, is a basic characteristic of geometric hashing.

Unlike the first algorithm which is based on a "connectionist" view of the geometric hashing method, the second algorithm makes use of the parallel machine as a source of "intelligent memory." The algorithm is inspired by the method of inverse indexing [87], and is data parallel over *combinations* of small subsets of model features.

The main characteristic of the connectionist algorithm is that the various $(model, basis)$ combinations *are* the data items; these combinations are grouped by means of the hash table according to the invariant tuples that they generate. On the other hand, in the broadcast approach, the data items are the invariants generated by the $(model, basis)$ combinations, and they can be grouped based on the feature combinations that generate these combinations.

Again, the database is assumed to contain $M$ models, and each model $m$ has associated with it a set of features, $\mathcal{F}_m = \{(x_{m,k}, y_{m,k})\}_{k=1}^{n}$. Without loss of generality, the coordinates of the features of all models are assumed to reside in the local memory of the host computer. The number of features extracted from the input image is $S$. Furthermore, we will make no assumptions with regard to the transformation that the database models are allowed to undergo; the cardinality of the basis tuple will be a parameter, and equal to $c$. Finally, the availability of $\Theta(Mn^{c+1})$ PE's is assumed.

### 3.6.1 The Data Structure

The main idea behind the algorithm is the following. Let us assume that we have formed a basis tuple, $\mathcal{B}$, by selecting a set of $c$ features from model $m$. The coordinates of each of the remaining $n - c$ features of model $m$, in the coordinate system defined by $\mathcal{B}$, will need to be computed; this will generate a total of

$n - c$ coordinate pairs. In other words, there will be one coordinate pair for every subset of model features comprising the selected basis $\mathcal{B}$ and each of the remaining $n - c$ features of $m$. One could conceivably dedicate one PE for each such combination, requiring $\binom{n}{c}(n-c)c!$ PE's per database model; each such PE holds the corresponding coordinate pair.

Evidently, a data structure that is different from the hash table data structure of section 3.5 is required. The data can be regarded as a collection of records of the form $(m, (i_1, i_2, \ldots, i_c), k, (x, y))$, where $(x, y)$ is the hash location to which the $k$-th feature of model $m$ maps under the basis tuple $\mathcal{B} = (i_1, i_2, \ldots, i_c)$. This information can be organized in a $(c + 2)$-dimensional table indexed by $(m, (i_1, i_2, \ldots, i_c), k)$. Recall that $i_1, i_2, \ldots, i_c, k$ are integers between 1 and $n$; $m$ is an integer between 1 and $M$. Clearly, not all of the $M\binom{n}{c}(n-c)c!$ locations will be used: for example, the entries of the table corresponding to *degenerate* bases will be empty. Similarly, the table locations corresponding to feature subsets where the $k$-th model feature has also been used to form $\mathcal{B}$ will also be empty. Then self-index of a table location suffices to recover the corresponding $(m, (i_1, i_2, \ldots, i_c), k)$ information. In the sequel, we will refer to this data structure as the *hash-function table*.

### 3.6.2 Hash-location Broadcast: Preprocessing Phase

During the preprocessing phase the hash-function table is constructed for the set of $M$ models. Three sets of virtual processors participate in this phase: the model feature set $V_1$, the $(c+1)$-product set $V_2$, and the hash-function table set $V_3$. Each processor of $V_3$ is associated with exactly one table location. This phase consists of three stages.

**Preprocessing Phase**

For each model $m$ do:

**Stage 1:** The host relays the coordinates of the $m$-th model's features to the $n$ PE's of set $V_1$: the $i$-th element of set $\mathcal{F}_m$ will reside in the local memory of the $i$-th PE of set $V_1$.

**Stage 2:** The $(c+1)$-product $(\mathcal{F}_m)^{c+1}$ is computed using the $p$-product algorithm described in section 3.4. Each of the $n^{c+1}$ processors of set $V_2$ now contains a $(c+1)$-tuple of the form $\left[(x_{i_1}, y_{i_1}), (x_{i_2}, y_{i_2}), \ldots, \left(x_{i_{(c+1)}}, y_{i_{(c+1)}}\right)\right] \in (\mathcal{F}_m)^{c+1}$. The first $c$ points of each such tuple define an ordered basis and thus a coordinate system. Some of those bases are formed by repeating the same feature point, and thus are degenerate; the corresponding virtual processors will be disabled and will not participate in the rest of the computation. Additionally, those processors in charge of tuples where the $(c+1)$-st point coincides with one of the points forming the basis will be disabled as well. The remaining processors proceed to compute the coordinates of the $(c+1)$-st point in the coordinate system defined by the basis.

**Stage 3:** Each processor of $V_2$ with information destined for a certain table location, sends a tuple of the form $(m, (i_1, i_2, \ldots, i_c), k, (x, y))$ to that location. Since the destinations of these messages are pairwise distinct, no collisions could possibly occur during this stage.

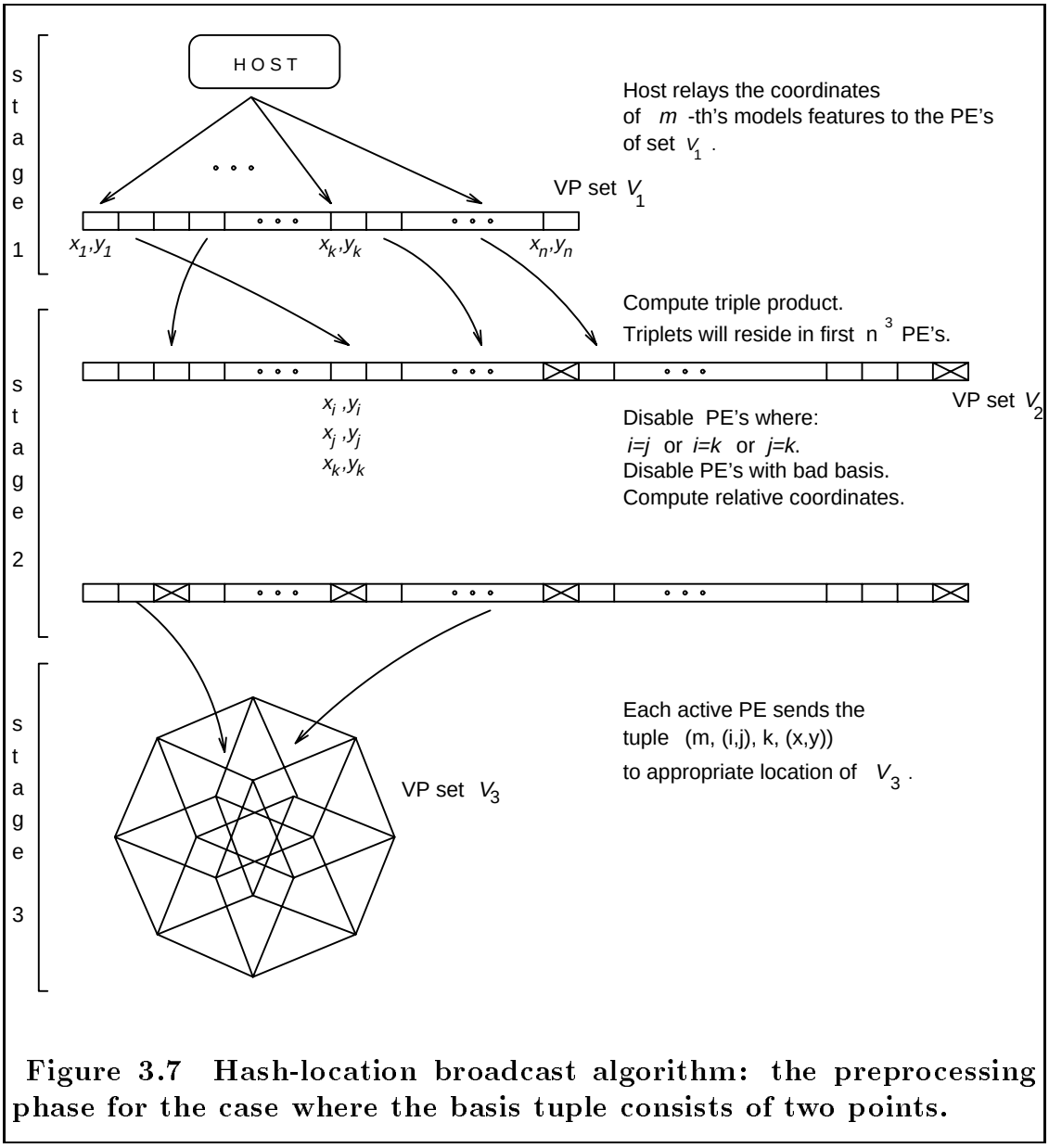


**Figure 3.7 Hash-location broadcast algorithm: the preprocessing phase for the case where the basis tuple consists of two points.**

Figure 3.7 graphically depicts the preprocessing phase for the case where the basis tuple consists of two points.

### 3.6.3 Hash-location Broadcast: Recognition Phase

Two virtual processor sets participate in this phase: the feature coordinate set $V_1$, and the hash-function table set $V_2$.

We assume that the coordinates of the features that were extracted from the input image reside in the local memory of the $S$ processors of the set $V_1$: the coordinates of the $i$-th image feature reside in the memory of the $i$-th processor of $V_1$. The hash-function table is again preloaded from storage into the local memory of the processors of $V_2$. This phase proceeds in three stages.

**Recognition Phase**

**Stage 1:** The host selects a basis tuple (a set of $c$ image features) and relays its coordinates to the $S$ processors of $V_1$. With the exception of the $c$ processors whose interest points form the selected basis, each processor in $V_1$ combines the coordinates of the feature stored locally with the broadcast tuple to compute the feature's coordinates in the system defined by the basis. These operations involve minimal data movement and thus are extremely fast.

**Stage 2:** In the second stage, the data from $S - c$ processors in $V_1$ are successively broadcast to all the processors of the set $V_2$. Each broadcast contains a coordinate of the form $(u, v)$, and gives a location in the hash table where a vote should be tallied. Each processor in $V_2$, indexed by $(m, (i_1, i_2, \ldots, i_c), k)$ with $(i_1, i_2, \ldots, i_k)$ corresponding to a valid basis tuple for model $m$, contains a hash location (coordinate pair) which the processor can compare against $(u, v)$. If the

two locations are sufficiently close together, then the corresponding table location records a hit indicating a vote for model $m$ and basis $(i_1, i_2, \ldots, i_k)$. We will see in chapter 7, that an extremely useful modification permits the use of weighted voting for model-basis tuples determined according to the relative proximity of $(u, v)$ to $(x, y)$. This vote originates from the particular feature (among the $S - c$ extracted from the input image) whose coordinates in the frame of the selected basis are being broadcast. The tallying of votes continues by accumulating hits in each hash-function table location; each of the $S - c$ broadcasts generates either one or no hits at any table location.

**Stage 3:** Upon completion of the tallying step, a third stage is invoked; using a segmented parallel-sum operation, we add the votes over $k$ among locations $(m, (i_1, i_2, \ldots, i_c), k)$. The result is the total number of votes that the model $m$ and the basis $(i_1, i_2, \ldots, i_c)$ obtain for the given scene and basis selection. Finally, a global-max among the processors associated with the locations holding the sum of votes is used to determine the winning $(model, basis)$ combination. A final verification step may be added to determine the quality of the match.

---

Figure 3.8 details the recognition phase for the case where the basis tuple consists of two points.

An asymptotically faster alternative to the Stages 2 and 3 described above also exists. Strictly speaking, each one of the $S - c$ broadcasts will require $\mathcal{O}(\log(Mn^{c+1}))$ time, since there are $Mn^{c+1}$ processors in the $V_2$ data set. However, assuming the existence of $S$ storage locations in each processor of set $V_2$, the

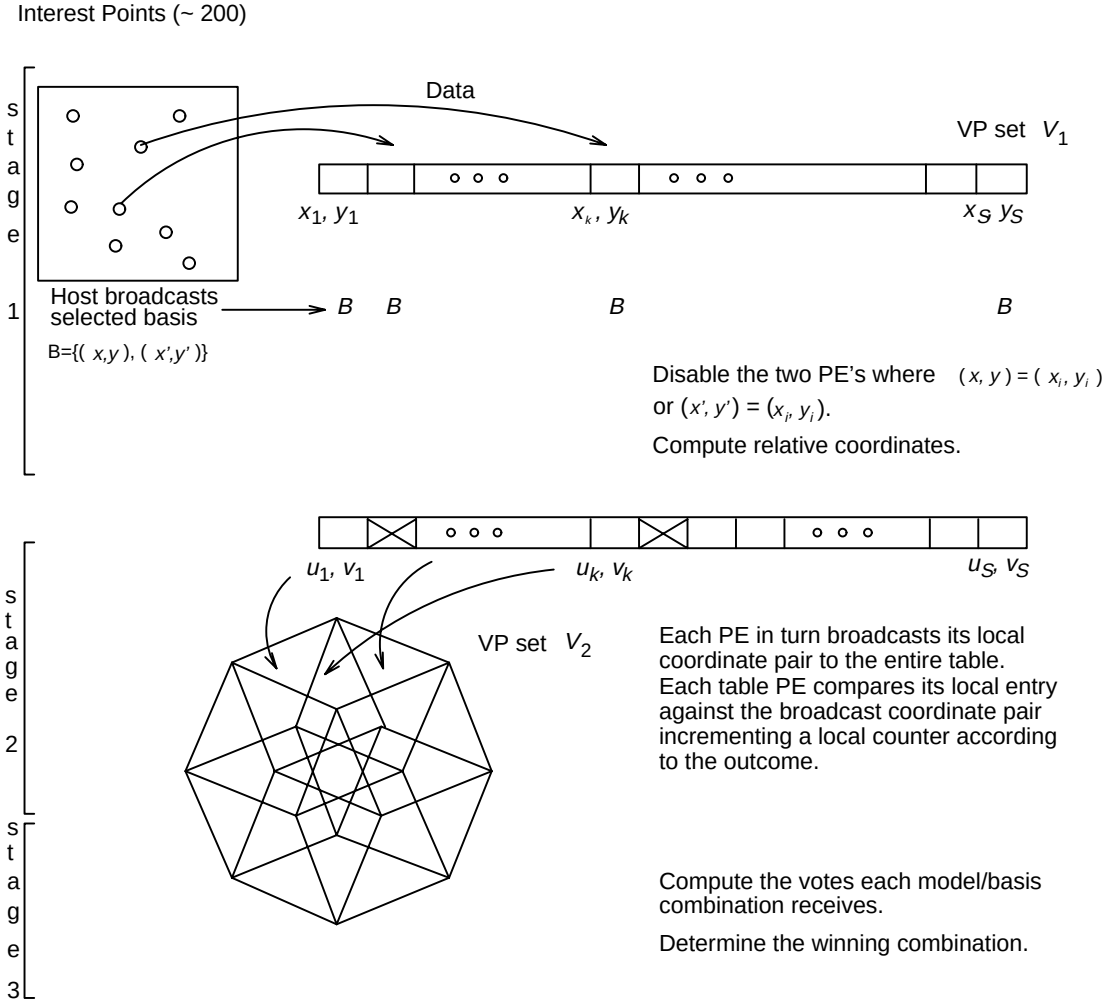


**Figure 3.8 The recognition phase of the parallel hash-location broadcast algorithm, for the case where the basis tuple consists of two points.**

theoretical complexity can be decreased. This can be accomplished as follows.

Assume for simplicity that $S = n^c$. Then all $S - c$ broadcasts can be done simultaneously, by having each of the $S - c$ active processors in $V_1$ send its data to a *unique* processor in a $c$-dimensional $(n \times n \times \ldots \times n)$ *slice* of the $(c + 2)$-dimensional data set $V_2$. This routing can be completed in time $\mathcal{O}(\log n)$. This slice of data can subsequently be spread to the rest of $V_2$, in parallel slices, requiring no more than $\mathcal{O}(\log(Mn))$ time. Observe that after this spread the entire set of the computed coordinate pairs is distributed among the $n^c$ processors of a slice, one coordinate pair per processor. Exactly $c$ processors in each slice will be empty. The processors within a slice can now exchange their data in such a way that the entire list of computed coordinate pairs becomes available to every single one of them. This can be simply achieved by a recursive doubling procedure which communicates data between pairs of processors, and forms lists of coordinate pairs. It must be noted though that the entries of those lists will not necessarily appear in the same order in each processor. This recursive doubling procedure can be completed in $\mathcal{O}(S)$ time. Total time complexities are summarized in the next subsection.

### 3.6.4 Time Complexity

For the time complexity of the two phases, we assume that $M\binom{n}{c}(n-c)c!$ PE's are available. The interconnection network is a hypercube, and the concurrent-read-exclusive-write (CREW) model of computation is adopted. We further assume that $M > n$, $M > S$, and $Mn^{c+1} \gg B$.

**Preprocessing Phase:** The complexity of this phase is dominated by the third stage. During that stage, each processor of set $V_1$ sends a message of the form $(m, (i_1, i_2, \ldots, i_c), k, (x, y))$ to a processor in $V_2$. The destinations of all these messages are pairwise distinct; thus no collision-resolution protocols will be required. As a result, the time complexity of this stage is $\mathcal{O}(\log(Mn))$ per database model. The time complexity of the preprocessing phase is thus $\mathcal{O}(M \log(Mn))$. The preprocessing phase is very expensive, even when carried out in parallel, but again, this is of no concern since it is executed off-line.

**Recognition Phase:** The time complexity of the recognition phase, per broadcast basis tuple, is dominated by the second stage. Indeed, the time complexity of the first stage is $\mathcal{O}(\log S)$. The second stage, using the data-spreading trick described at the end of the previous section results in time complexity which is no worse than $\mathcal{O}(S + \log(Mn))$. This is also the complexity of the recognition phase for the hash-location broadcast algorithm.

## 3.7 Implementation Details

In this section, we present in detail the actual implementations of the two algorithms already described. Both algorithms were implemented on a hypercube-based SIMD Connection Machine.

Implementing carefully-crafted parallel algorithms on existing architectures frequently involves more compromises than one might expect. In our case, the two algorithms that we have described have assumed the existence of $\Theta(Mn^{c+1})$ PE's. For $M = 1024$, $n = 16$, and the similarity transformation $(c = 2)$ this would entail a 4Meg-processor machine. Although there exist $64K$-processor Connection

Machines, it is more usual to have access to a $32K$- or $16K$-processor model, and to do the prototyping on an $8K$-processor model. Conceivably, one could make use of the Connection Machine software facilities to simulate a larger parallel machine by mapping multiple processors to each physical processor, which then execute in round-robin fashion. Unfortunately, the overhead involved in the mapping makes the implied "virtual processor ratio" of 512 impractical. Accordingly, we must modify the algorithms somewhat.

**Connectionist Algorithm.** Rather than associating a separate processor with each hash entry, we can store the entire list of entries for a hash bin in the local memory of a *single* processor. In order to implement the algorithm, all the entries that hash into the same bin are stored contiguously in a *single* physical processor memory rather than associated with distinct physical processors. More specifically, each hash table bin is mapped onto a physical processor; this processor maintains locally a list of all the $(m, (i_1, i_2, \ldots, i_c))$ entries hashing into the corresponding hash table bin. The required number of processors drops to $B$, the number of desired hash bins.

The preprocessing phase is now far less efficient, due to the need for random access to local memory as entries are appended to the lists. Clearly, the lengths of the lists will vary over the hash table, and some of them will be empty. For a given database of models, the typical occupancies for uniformly quantized hash bins are non-uniform (see Figure 3.9). Provided that no single list is exorbitantly long, memory requirements are not a problem. The collisions that are likely to occur during this phase are resolved using a simple protocol based on "locks."

During recognition, the entries in the hash bins that receive votes must be

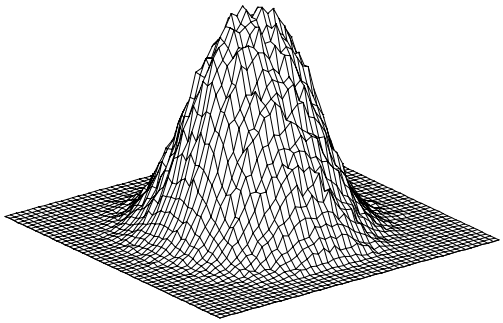

**Figure 3.9 Hash bin occupancy for a typical database; the height is proportional to the length of the corresponding hash bin list.**

histogramed (i.e., counted) with the multiplicity of the number of votes that each hash bin receives. For $M$ models each having $n$ points and a basis tuple comprising $c$ features, each hash entry need only be $\lceil \log M + c \log n \rceil$-bits long: $\log M$ bits for the model index, and $\log n$ bits for the index of each of the $c$ basis members. Rather than histograming by sorting, we opt, as already indicated, for a message passing strategy (additive writes): we set up $2^{\lceil \log M + c \log n \rceil}$ buckets (which results in a virtual processor ratio of $2^{\lceil \log M + c \log n \rceil - 13}$ on the $8K$-processor machine), one bucket for each $(m, (i_1, i_2, \ldots, i_c))$ combination.

The processors in charge of those hash bins that received one or more votes, scan their local lists and cast a vote for each entry $(m, (i_1, i_2, \ldots, i_c))$ encountered by sending a message to the corresponding histogram bucket. The time needed for the list traversal is clearly dominated by the longest such list.

This histograming process currently accounts for 99% of the execution time of the recognition phase. Clearly, efficiencies in histograming will very much improve

the performance of the implementation. In particular, the use of our radix-sort based method of Section 3.4.2 is expected to considerably reduce processing times. Further improvements in efficiency can be achieved by requantization of the hash space, and the use of symmetries, and will be examined in more detail in chapter 5.

**Hash-location Broadcast Algorithm.** In order to reduce the virtual processor ratio for set $V_2$ in the implementation of the second algorithm, we have chosen to assign one processor to each index $(m, (i_1, i_2, \ldots, i_c))$, and store contiguously in that processor's local memory the $n$ entries associated with $k = 1, 2, \ldots, n$. This in effect collapses one of the dimensions of the hash-function table. Consequently, during the construction of the hash-function table, precisely $n - c$ processors attempt to deposit a distinct tuple at the same destination. Use of "locks" provides the necessary serialization.

During recognition, the $S$ processors of $V_1$ rank themselves using either the index of the locally available image feature, or their own address: either approach to performing the ranking involves no data movement and is therefore extremely fast. After having computed the relative coordinates of the local feature in the frame determined by the broadcast basis, each of $S - c$ processors of $V_1$ in turn broadcasts the computed coordinates to all of the processors of the hash-function table: each processor of $V_2$ compares the broadcast location with each of the $n - c$ locations stored in its local memory, updating a local counter if necessary. A total of $(S - c)(n - c)$ comparisons will be required per basis selection. When all image features have been exhausted, a global-max operation on the values of the local counters recovers the winning $(m, (i_1, i_2, \ldots, i_c))$ combination. Computational efficiencies can again be achieved by making use of symmetries, and will

be examined in more detail in chapter 5.

**Languages.** Although a number of special-purpose parallel languages have been developed for the Connection Machine, we found that *C* code running on the host enhanced with system calls to the Connection Machine using its Paris package is the most suitable for our needs. The Paris package includes routines that give us the greatest level of control over the machine. For the broadcast-based algorithm, just about any language would suffice, and the Paris primitives represent a fast development path.

## 3.8 Implementation Results / Scalability

Models (dot patterns) of 16 points each were generated using either a uniform distribution over a region, or a Gaussian distribution. After generating 1024 such models, scenes were constructed of approximately 200 points, with a single model embedded in the scene, translated, rotated, and scaled. Noise was added to the scene points (through quantization round-off error).

In both implementations, the front end randomly selects a pair of scene points (a probe) as the basis to be used for possible recognition. For the connectionist algorithm, a probe takes 5.05 seconds on an $8K$-processor machine, dropping to 0.80 seconds on a $32K$-processor machine (see the plots in Figure 3.10). The employed hash table contained 86.6% of the total number of hash entries. It can be seen from the plots that the connectionist algorithm is operating in a roughly linear regime – i.e., we are achieving linear speedup due to the heavy loading. In fact, as the number of processors increases, reduced contention in the routing algorithm gives us, in some cases, an apparent extra boost; such improvements

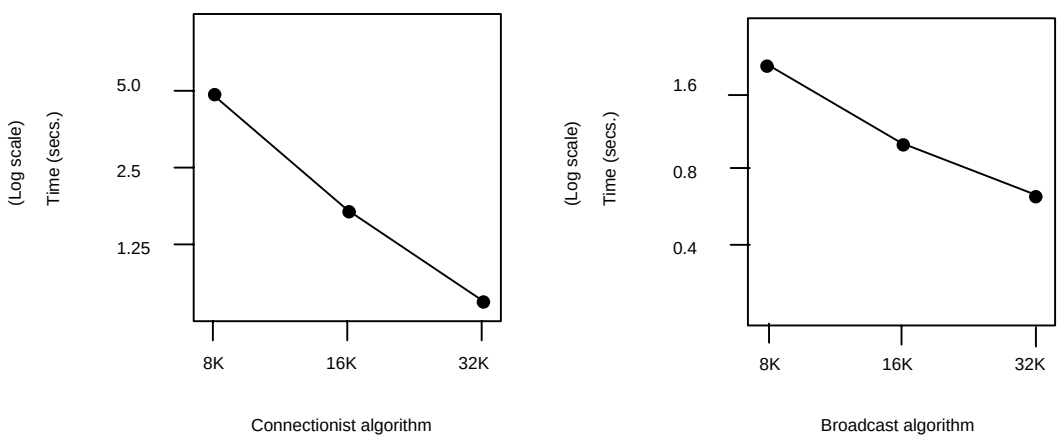


**Figure 3.10 Average time required for a single basis probe, as a function of the number of processors in the Connection Machine. The database contains 1024 models of 16 points each, and the scenes contain 200 points.**

would not continue forever.

In the broadcast algorithm, the $8K$-processor machine processes a probe at a rate of 10 msec per scene point, i.e., approximately 2.0 seconds for a probe using a two hundred point scene. The employed hash-function table contained 100% of the total number of hash entries for the given database. Experiments with a $16K$- and $32K$-processor model indicate nearly linear increases in speed (see Figure 3.10), so that a $64K$-processor machine should be able to perform a probe in about 300 milliseconds.

By way of comparison, both algorithms are easily coded on a typical high-performance workstation. Performance results are highly dependent on disk access rates and available memory, but we have seen probe times of roughly 35 seconds for the equivalent of the hash-location broadcast algorithm on a SUN Sparcstation-2.

# Chapter 4

# Distributions of Invariants

In this chapter, we examine the issue of index distribution over the space of invariants. In particular, we derive precise as well as approximate formulas and qualitative results for a number of transformation and feature distribution combinations.

Soon after the conception of the geometric hashing technique, researchers discovered that one of the main characteristics of the method was the non-uniform distribution of indices over the space of invariants [21,25,89]. This non-uniformity, which appears to be an endemic of all indexing-based approaches to object recognition, typically poses problems. A number of heuristics were invented in order to alleviate those problems [25], but the results were not particularly promising.

In our study, we will concentrate on the rigid, similarity and affine transformations. We will assume that the model point features are generated by either a Gaussian random process of standard deviation $\sigma$, or a process that is Uniform over the unit disc or the unit square.

As it will be demonstrated in chapter 5, the knowledge of those distributions is particularly important; indeed, it allows for a number of performance enhance-

ments in the implementations of the algorithms described in chapter 3. Also, the knowledge of those distributions proves instrumental in the development of a Bayesian approach to model matching with geometric hashing (see chapter 7).

## 4.1 Rigid Transformation

We begin our study of the distribution of indices over the space of invariants with the rigid transformation: the database models are allowed to undergo rotation and translation only.

Let $\mathbf{p}_{\mu_1}$, $\mathbf{p}_{\mu_2}$, and $\mathbf{p}$ be the position vectors of three point features belonging to model $m$. Then, the tuple $(u, v)$ satisfying Eqn. 2.1 is unique, and invariant under rigid transformations.

Let us further assume that the point features $\mathbf{p}_{\mu_1}$, $\mathbf{p}_{\mu_2}$, and $\mathbf{p}$ are generated by the Gaussian random process $\mathcal{N}(\mathbf{0}, \left(\begin{smallmatrix} \sigma & 0 \\ 0 & \sigma \end{smallmatrix}\right))$, i.e. a two-dimensional Gaussian process with mean value $(0, 0)$ and covariance matrix $\left(\begin{smallmatrix} \sigma & 0 \\ 0 & \sigma \end{smallmatrix}\right)$. From standard probability theory, we know that the joint distribution $f(u, v)$ of $u$ and $v$ is given by the expression

$$\int_{\mathbb{R}^4} f(x(u,v), y(u,v))\, f(x_{\mu_1}, y_{\mu_1})\, f(x_{\mu_2}, y_{\mu_2}) \mid J \mid^{-1} dx_{\mu_1} dx_{\mu_2} dy_{\mu_1} dy_{\mu_2}, \qquad (4.1)$$

where $J$ is the Jacobian of the transformation. Evaluation of this integral yields the following result for the distribution of indices over the space of invariants

$$\boxed{f(u,v) = \frac{1}{3\,\pi\,\sigma^2} \cdot e^{-\left(\frac{u^2+v^2}{3\sigma^2}\right)}}. \qquad (4.2)$$

Figure 4.1 shows the distribution over the space of invariants, and several of its contours.

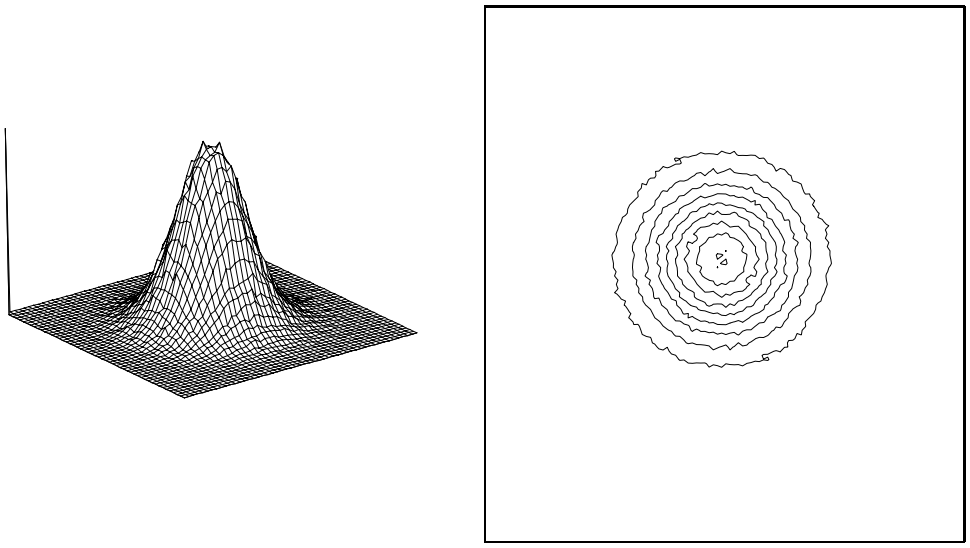

**Figure 4.1 The distribution over the space of invariants, and several of its contours for the case of point features that are generated by the Gaussian process $\mathcal{N}(\mathbf{0}, \left(\begin{smallmatrix} \sigma & 0 \\ 0 & \sigma \end{smallmatrix}\right))$. The allowed transformation is rigid.**

For the case where the point features $\mathbf{p}_{\mu_1}$, $\mathbf{p}_{\mu_2}$, and $\mathbf{p}$ are uniformly distributed over the unit disc, an evaluation of the above integral proves too difficult to obtain analytically. Consequently, non-linear parameter fitting was exploited. We applied the method of Levenberg-Marquardt [75] to synthetically generated data, and found that, in this case, the distribution of indices over the space of invariants can be approximated well by

$$\boxed{f(u,v) = \frac{1}{\left(\left(4.7\,u^2 + 3.9\,v^2\right)^2 + 36.7\right)^2}}\,. \tag{4.3}$$

The distribution of synthetically generated indices over the space of invariants, and several of its contours, are shown in Figure 4.2.

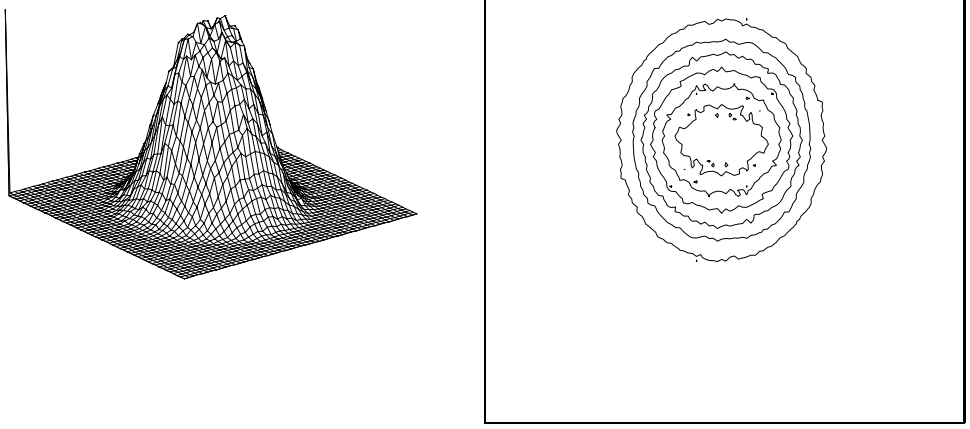

**Figure 4.2 The distribution over the space of invariants, and several of its contours for the case of point features that are uniformly distributed over the unit disc. The allowed transformation is rigid.**

## 4.2 Similarity Transformation

We next examine the distribution of indices over the space of invariants for the case of the similarity transformation: the models in the database are allowed to undergo rotation, translation and scaling.

If $\mathbf{p}_{\mu_1}$, $\mathbf{p}_{\mu_2}$, and $\mathbf{p}$ are the position vectors of three point features belonging to model $m$, then the tuple $(u, v)$ satisfying Eqn. 2.2 is unique, and invariant under similarity transformations.

If we further assume that the point features $\mathbf{p}_{\mu_1}$, $\mathbf{p}_{\mu_2}$, and $\mathbf{p}$ are generated by the Gaussian random process $\mathcal{N}(\mathbf{0}, \left(\begin{smallmatrix} \sigma & 0 \\ 0 & \sigma \end{smallmatrix}\right))$, then, evaluation of the integral in expression 4.1 yields

$$\boxed{f(u,v) = \frac{12}{\pi} \cdot \frac{1}{\left(4\left(u^2+v^2\right)+3\right)^2}} \,. \tag{4.4}$$

It is worth noting that the last expression is independent of the value of the standard deviation of the Gaussian process generating the point features. The distribution over the space of invariants, and several of its contours are shown in Figure 4.3.

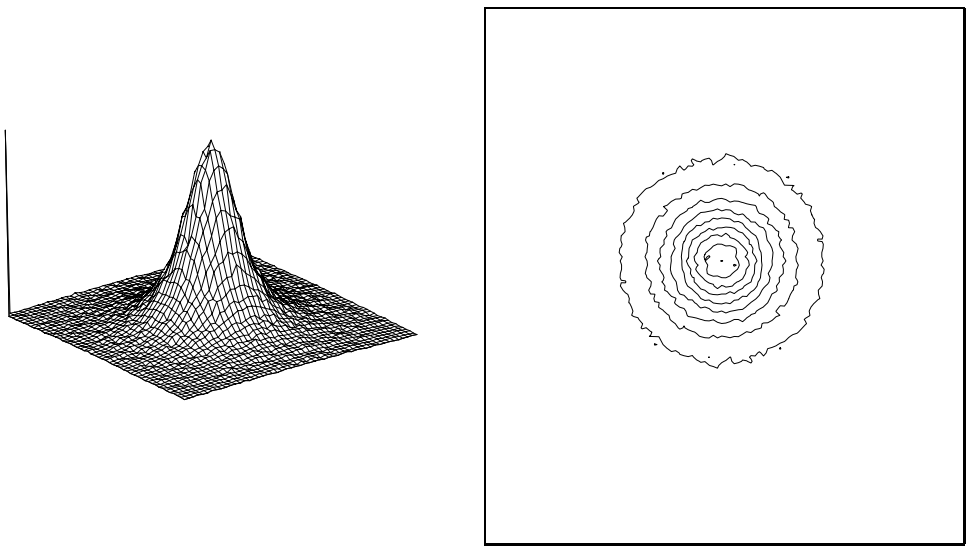

**Figure 4.3 The distribution over the space of invariants, and several of its contours for the case of point features that are generated by the Gaussian process $\mathcal{N}(\mathbf{0}, \left(\begin{smallmatrix} \sigma & 0 \\ 0 & \sigma \end{smallmatrix}\right))$. The allowed transformation is similarity.**

A derivation of the expression for the distribution of invariants in the case where the point features are uniformly distributed over either the unit disc, or the unit square has proven intractable.

## 4.3 Affine Transformation

The case where the patterns of point features corresponding to the different models can undergo a general linear (affine) transformation is slightly different. An

affine transformation of any such pattern will be uniquely defined by the transformation of three, instead of two, points.

Assume that $\mathbf{p}_{\mu_1}$, $\mathbf{p}_{\mu_2}$, and $\mathbf{p}_{\mu_3}$ are the position vectors of three point features belonging to model $m$. Then the vectors $\mathbf{p}_{\mu_2} - \mathbf{p}_{\mu_1}$ and $\mathbf{p}_{\mu_3} - \mathbf{p}_{\mu_1}$ form a skewed basis, and thus a skewed coordinate system $Oxy$. Any other point $\mathbf{p}$ of model $m$ can be represented in this basis as

$$\mathbf{p} - \mathbf{p}_0 = u\,(\mathbf{p}_{\mu_2} - \mathbf{p}_{\mu_1}) + v\,(\mathbf{p}_{\mu_3} - \mathbf{p}_{\mu_1}) \tag{4.5}$$

where $\mathbf{p_0} = \alpha\mathbf{p}_{\mu_1} + \beta\mathbf{p}_{\mu_2} + \gamma\mathbf{p}_{\mu_3}$ is the position vector of the center of the skewed coordinate system. The reason for our expressing the coordinates of the latter point as a function of $\mathbf{p}_{\mu_1}$, $\mathbf{p}_{\mu_2}$, and $\mathbf{p}_{\mu_3}$ will become evident shortly.

Let us assume that the point features $\mathbf{p}_{\mu_1}$, $\mathbf{p}_{\mu_2}$, $\mathbf{p}_{\mu_3}$, and $\mathbf{p}$ are generated by the Gaussian random process $\mathcal{N}(\mathbf{0}, \left(\begin{smallmatrix}\sigma & 0\\ 0 & \sigma\end{smallmatrix}\right))$, i.e. a two-dimensional Gaussian process with mean value $(0,0)$ and covariance matrix $\left(\begin{smallmatrix}\sigma & 0\\ 0 & \sigma\end{smallmatrix}\right)$. Then from standard probability theory, we know that the joint distribution $f(u,v)$ of $u$ and $v$ is given by the expression

$$\int_{\mathbb{R}^6} f(x(u,v), y(u,v))\; f(x_{\mu_1}, y_{\mu_1})\; f(x_{\mu_2}, y_{\mu_2})\;\; f(x_{\mu_3}, y_{\mu_3})\;\; \mid J \mid^{-1} \\ dx_{\mu_1} dx_{\mu_2} dx_{\mu_3} dy_{\mu_1} dy_{\mu_2} dy_{\mu_3}, \tag{4.6}$$

where $J$ is the Jacobian of the transformation. Evaluation of the latter integral yields the following result for the distribution of indices over the space of invariants

$$\boxed{f(u,v) = \frac{\mathcal{C}}{\left(4u^2 + 4v^2 + 4uv + 4\,(\beta - \alpha)\,u + 4\,(\gamma - \alpha)\,v + 2\left(\alpha^2 + \beta^2 + \gamma^2 + 1\right)\right)^{\frac{3}{2}}}} \tag{4.7}$$

where $\mathcal{C}$ is a normalization constant that makes $f(u,v)$ a probability density function.

Several observations regarding the distribution over the space of invariants can be made. In particular, from this last equation we can see that the contours of the distribution are second order curves of the *elliptic* type. These curves have two axes of symmetry: one of the axes is at 45 degrees to the horizontal whereas the second is perpendicular to the first, and this is *independent* of the choice of $\alpha$, $\beta$, and $\gamma$.

In addition to the axial symmetry, the curves also have a *center of symmetry*, which is located at

$$\boxed{(u_{cs}, v_{cs}) = \left(\frac{\alpha - 2\beta + \gamma}{3}, \frac{\alpha + \beta - 2\gamma}{3}\right)}. \tag{4.8}$$

The location of the center of symmetry, which happens to coincide with the location of the peak of the distribution, allows the derivation of a natural constraint on the pivot selection. If we require that the center of symmetry coincide with the center of the coordinate system in the space of invariants, i.e. that

$$u_{cs} = 0 \quad \wedge \quad v_{cs} = 0, \tag{4.9}$$

we conclude from Eqn. 4.8 that it must hold that

$$\boxed{\alpha = \beta \quad \wedge \quad \beta = \gamma}. \tag{4.10}$$

The choice for the location of the center of the skewed coordinate system should be clear. By setting $\alpha = \beta = \gamma = 1/3$, we effectively require that the center coincide with the *barycenter* of the triangle defined by $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$, a point that is always well-defined since $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ are assumed to be in general position.[1]

[1] A similar result holds for the case of the rigid and similarity transformations. It can be easily shown that the natural choice for $\mathbf{p}_0$ in these cases is the midpoint between the points defining the basis tuple.

For this selection of $\alpha$, $\beta$, and $\gamma$, the expression of the distribution of the hash entries over the hash table becomes

$$\boxed{f(u,v) = \frac{2\sqrt{2}}{\pi} \cdot \frac{1}{(4u^2 + 4v^2 + 4uv + 8/3)^{\frac{3}{2}}}}. \tag{4.11}$$

Note that, as was the case with the similarity transformation, the probability density function is *independent* of the standard deviation of the Gaussian process generating the feature points. Figure 4.4 shows the distribution over the space of invariants, and several of its contours.

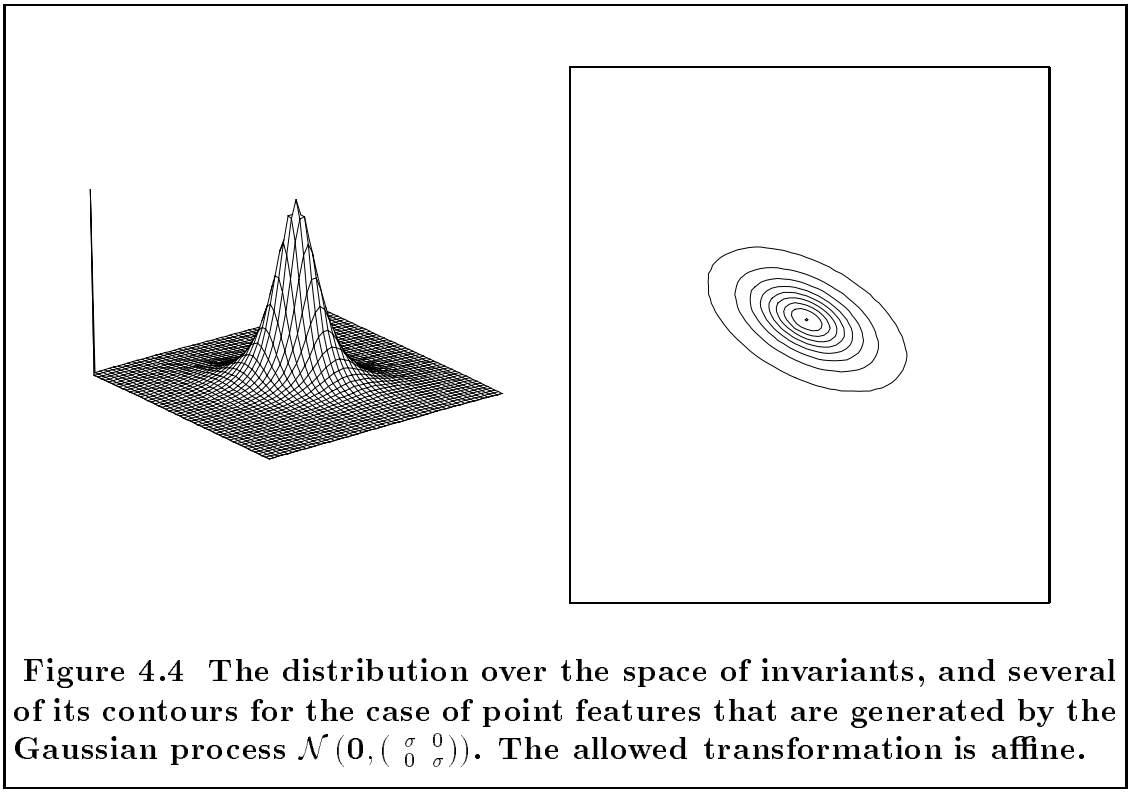

**Figure 4.4 The distribution over the space of invariants, and several of its contours for the case of point features that are generated by the Gaussian process $\mathcal{N}(\mathbf{0}, \left(\begin{smallmatrix} \sigma & 0 \\ 0 & \sigma \end{smallmatrix}\right))$. The allowed transformation is affine.**

We next extend our analysis to the case where the model point features have a uniform distribution over a convex domain, for example the unit disc or the unit square. In principle, one could attempt to evaluate the integral in the expression 4.6. Unfortunately, the evaluation of this multiple integral was too difficult

to obtain analytically. Further, use of non-linear parameter fitting did not prove helpful. However, as we will see below, a *qualitative* description of the distribution is possible.

With regards to the selection of the position of the center of the skewed coordinate system, and in spite of the fact that the barycenter is a natural selection, it is not at all clear whether the above analysis carries over to the case of uniformly distributed model features. Consequently, in what follows we assume that the pivot coincides with $\mathbf{p}_1$ as was originally described in [61].

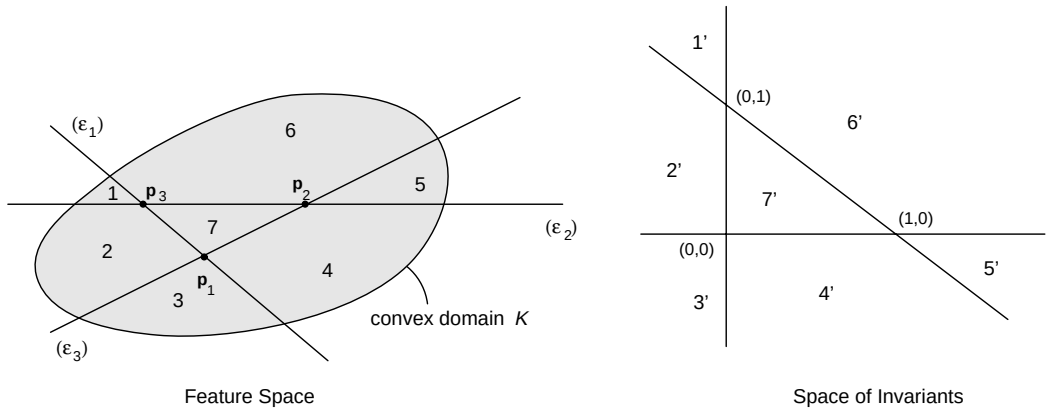


**Figure 4.5 Correspondence between feature and hash space regions: if p lies in the region of the feature space marked i, the computed invariant tuple will lie in the region i' of the space of invariants.**

Let $\mathbf{p}_1, \mathbf{p}_2$, and $\mathbf{p}_3$ be three points of $\mathbb{R}^2$ in general position (Figure 4.5). Lines $(\epsilon_1), (\epsilon_2)$, and $(\epsilon_3)$ divide the feature and hash spaces into seven regions: if the point $\mathbf{p}$ lies in the region of the feature space marked $i$, $i \in \{1, 2, \ldots, 7\}$, the generated tuple invariant will lie in the region of the space of invariants marked $i'$. Notice that if $\mathbf{p}$ lies in any of the odd-numbered regions the quadrangle formed by the four points will be reentrant, whereas if $\mathbf{p}$ lies in any of the even-

numbered regions the quadrangle will be convex. In order to *qualitatively* describe the distribution over the hash table, we seek the probability that the generated invariant tuple will lie in a given quadrant of the space of invariants. To this end, we will make use of the answer to a famous problem from geometric probability: the so-called Sylvester's *Vierpunkt* problem[4]. This problem can be stated as follows: "given a convex domain $K$, find the probability that four points taken at random inside $K$ will form a reentrant quadrilateral." As it turns out, the answer to this problem varies with the shape of $K$; for regular polygons and circles/ellipses this probability is very close to 0.3.

Let $\Pr(C)$ (respectively $\Pr(R)$) denote the probability that the quadrilateral formed by $\mathbf{p}, \mathbf{p}_1, \mathbf{p}_2$ and $\mathbf{p}_3$ is *convex* (respectively *reentrant*). From Figure 4.5 and using a symmetry argument, we can see that

$$\begin{aligned}
\Pr(\mathbf{p} \text{ in } 2) &= \Pr(\mathbf{p} \text{ in } 4) = \Pr(\mathbf{p} \text{ in } 6) = \tfrac{1}{3}\Pr(C) \\
\Pr(\mathbf{p} \text{ in } 1) &= \Pr(\mathbf{p} \text{ in } 3) = \Pr(\mathbf{p} \text{ in } 5) = \Pr(\mathbf{p} \text{ in } 7) = \tfrac{1}{4}\Pr(R).
\end{aligned}$$

We can then evaluate the desired probabilities as follows:

$$\begin{aligned}
\Pr(\text{tuple in 1st quadrant}) &= \Pr(\text{tuple in } 6' \text{ or } 7') = \Pr(\mathbf{p} \text{ in } 6) + \Pr(\mathbf{p} \text{ in } 7) \\
&= \tfrac{1}{3}\Pr(C) + \tfrac{1}{4}\Pr(R) = \tfrac{1}{3}(1 - \Pr(R)) + \tfrac{1}{4}\Pr(R) \\
&= \tfrac{1}{3} - \tfrac{1}{12}\Pr(R)
\end{aligned}$$

Working in a similar manner,

$$\begin{aligned}
\Pr(\text{tuple in 2nd quadrant}) &= \tfrac{1}{3} - \tfrac{1}{12}\Pr(R) \\
\Pr(\text{tuple in 3rd quadrant}) &= \tfrac{1}{4}\Pr(R) \\
\Pr(\text{tuple in 4th quadrant}) &= \tfrac{1}{3} - \tfrac{1}{12}\Pr(R).
\end{aligned}$$

The value of $\Pr(R)$ in the above formulas is given by the solution to Sylvester's problem for the given convex domain $K$. In the case where $K$ is the unit disc,

$\Pr(R) = 35/(12\pi^2)$ and

$$
\begin{array}{lcl}
\Pr(\text{tuple in 1st quadrant}) & = & \frac{1}{3} - \frac{35}{144\pi^2} \\
\Pr(\text{tuple in 2nd quadrant}) & = & \frac{1}{3} - \frac{35}{144\pi^2} \\
\Pr(\text{tuple in 3rd quadrant}) & = & \frac{35}{48\pi^2} \\
\Pr(\text{tuple in 4th quadrant}) & = & \frac{1}{3} - \frac{35}{144\pi^2}
\end{array}
$$

In other words, the first, second, and fourth quadrants will each contain the same number of hash entries, whereas only a *very small* percentage of the entries (less than 8% of the total) will reside in the third quadrant! Consequently, the resulting distribution will have only one axis of symmetry. This is a rather unexpected result given the shape of the domain $K$ (unit disc). If $K$ is a square,

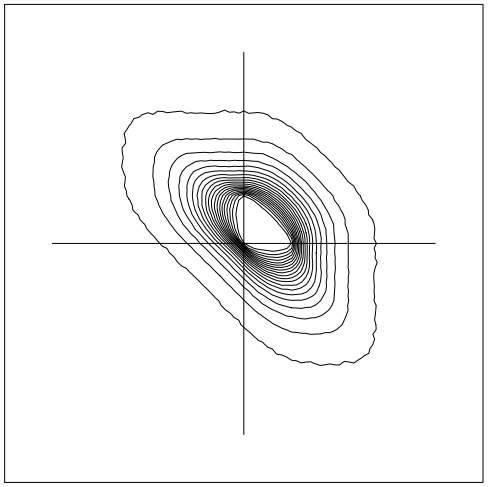

**Figure 4.6 Several of the contours of the hash table distribution. The model features are uniformly distributed over the unit disc, and the allowed transformation is affine.**

the value of $\Pr(R)$ is equal to 11/36: substitution of the latter value to the above

formulas gives

$$\begin{aligned}
\Pr(\text{tuple in 1st quadrant}) &= \tfrac{133}{432} \\
\Pr(\text{tuple in 2nd quadrant}) &= \tfrac{133}{432} \\
\Pr(\text{tuple in 3rd quadrant}) &= \tfrac{11}{144} \\
\Pr(\text{tuple in 4th quadrant}) &= \tfrac{133}{432}
\end{aligned}$$

In other words, the asymmetry still persists. Figure 4.6 shows several of the contours of the distribution over the hash table for the case where $K$ is the unit disc. This distribution was obtained by means of a Monte Carlo simulation. As can be seen, practically all of the hash entries are located in the first, second and fourth quadrants of the hash space, as predicted by the above analysis.

The above results also hold qualitatively for the case where the pivot coincides with the barycenter.

# Chapter 5

# Parallelism Revisited

In this chapter, we present a number of enhancements to the geometric hashing method. In particular, hash table equalization and exploitation of symmetries are developed specifically for the parallel algorithms. These techniques lead to substantial performance improvements and are also applicable to more general implementations of indexing-based object recognition methods.

As was indicated in Section 3.7, the non-uniform occupancy of the hash bins results in different lengths for the hash entry lists. Since the time needed to traverse the hash entry lists during the histograming phase of the algorithm is dominated by the longest such list, a uniform distribution of the entries over the hash table is desirable; a uniform distribution will reduce execution time and result in an efficient storage of the hash table data structure.

In addition to the efficiencies gained by rehashing, we may independently make use of certain symmetries in the storage pattern of hash entries in the hash table: exploitation of these symmetries results in further savings in computational and storage requirements.

## 5.1 Rehashing

We next describe a method to transform the coordinates of point locations so that the equispaced quantizer in the space of invariants yields an expected uniform distribution.

One must first determine the expected probability density $f(u, v)$ of the distribution of the untransformed coordinates (tuple of invariants). As we described in chapter 4, this can be done either by fitting a parametric model to synthetically generated data, or by calculating analytically the expected distribution based upon a model for the distribution of point features in the plane.

Once the probability density $f(u, v)$ is known, a transformation that maps the original distribution to the uniform distribution over a closed region (in particular, a rectangle) must be computed. This new mapping is effectively a hashing function

$$\mathbf{h}: \quad \mathbb{R}^2 \quad \rightarrow \quad \mathbb{R}^2,$$

and is used to evenly distribute the bin entries over a rectangular hash table. When the appropriate remapping function is applied to the invariants computed using either one of Eqns. 2.1, 2.2, or 2.3, then the equally-spaced hash bins in the remapped space have a uniform expected population. Henceforth, this function will be called a *rehashing* function.

**Case of Rigid Transformations.** In Section 4.1 we derived formulas for the distribution $f(u, v)$ of indices over the space of invariants for the case where the point features were either generated by the Gaussian random process $\mathcal{N}(\mathbf{0}, \left(\begin{smallmatrix} \sigma & 0 \\ 0 & \sigma \end{smallmatrix}\right))$, or, uniformly distributed over the unit disc.

We begin with the case of Gaussian distributed point features. The distribution of indices over the space of invariants was shown to be

$$f(u,v) = \frac{1}{3\,\pi\,\sigma^2}\; e^{-\dfrac{u^2+v^2}{3\sigma^2}}\;.$$

The expression can be rewritten in polar coordinates $(\rho,\theta)$ as follows

$$f(\rho,\theta) = \frac{1}{3\,\pi\,\sigma^2}\; e^{-\dfrac{\rho^2}{3\sigma^2}}, \quad \rho \in [0,\infty\ )\,,\ \ \theta \in [0,2\pi\ )\,.$$

Observing that

$$\int_0^{\rho_0}\int_0^{2\pi}\ \rho\; f(\rho,\theta)\; d\theta d\rho = \left(1 - e^{-\dfrac{\rho_0^2}{3\sigma^2}}\right),$$

we conclude that the rehashing function is given by

$$\boxed{\mathbf{h}(u,v) = \left(1 - e^{-\dfrac{u^2+v^2}{3\sigma^2}}\ ,\quad \mathrm{atan2}(v,u)\right)}\;. \tag{5.1}$$

The function atan2$(\cdot,\cdot)$ converts rectangular coordinates to polar coordinates by returning the phase in the interval $[-\pi,\pi]$. Figure 5.1 shows the result of hash table equalization for the case of rigid transformations, and several of the corresponding contours. The point features were generated synthetically by the Gaussian process $\mathcal{N}\,(\mathbf{0},(\begin{smallmatrix}\sigma & 0\\ 0 & \sigma\end{smallmatrix}))$.

When the model point features are uniformly distributed over the unit disc, the distribution of indices over the space of invariants can be well-approximated by the function

$$f(u,v) = \frac{1}{((4.7\,u^2 + 3.9\,v^2)^2 + 36.7)^2}.$$

Repeating the above analysis, we can show that the rehashing function in this case is

$$\boxed{\mathbf{h}(u,v) = \left(\frac{2}{\pi}\mathrm{atan}\left(\frac{(au)^2+\left(bv\right)^2}{c^2}\right) + \frac{2\left[(au)^2+\left(bv\right)^2\right]c^2}{\pi\left(\left[(au)^2+\left(bv\right)^2\right]^2+c^4\right)}, \mathrm{atan2}\left(v,u\right)\right)} \tag{5.2}$$

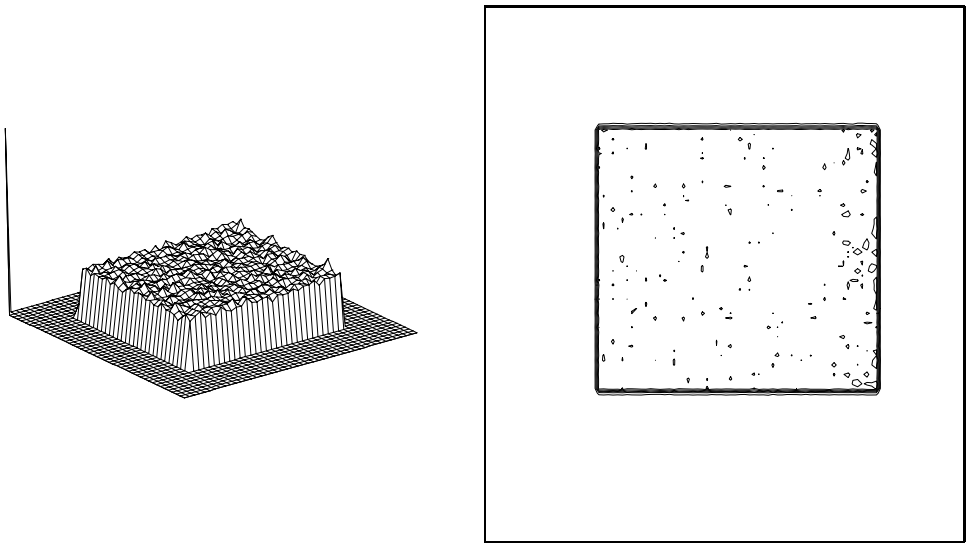

**Figure 5.1 Hash table equalization for the case of rigid transformations and point features generated by the Gaussian process $\mathcal{N}(\mathbf{0}, \left(\begin{smallmatrix}\sigma & 0\\ 0 & \sigma\end{smallmatrix}\right))$. Left: the expected distribution of remapped invariants. Right: several of the distribution's contours.**

where $a = 4.7$, $b = 3.9$, $c = (36.7)^{1/4}$, and $\mathrm{atan}(\cdot,\cdot)$ returns the arctangent of its argument in the interval $[-\pi/2, \pi/2]$. In Figure 5.2 we show the result of hash table equalization, and several of the corresponding contours. As can be seen, the remapping is very efficient.

**Case of Similarity Transformations.** For the case of the similarity transformation, we have determined the distribution of indices only when the point features are generated by a Gaussian process $\mathcal{N}(\mathbf{0}, \left(\begin{smallmatrix}\sigma & 0\\ 0 & \sigma\end{smallmatrix}\right))$. The expression for that distribution was shown to be independent of $\sigma$, and equal to

$$f(u,v) = \frac{12}{\pi} \frac{1}{\left(4\left(u^2+v^2\right)+3\right)^2}.$$

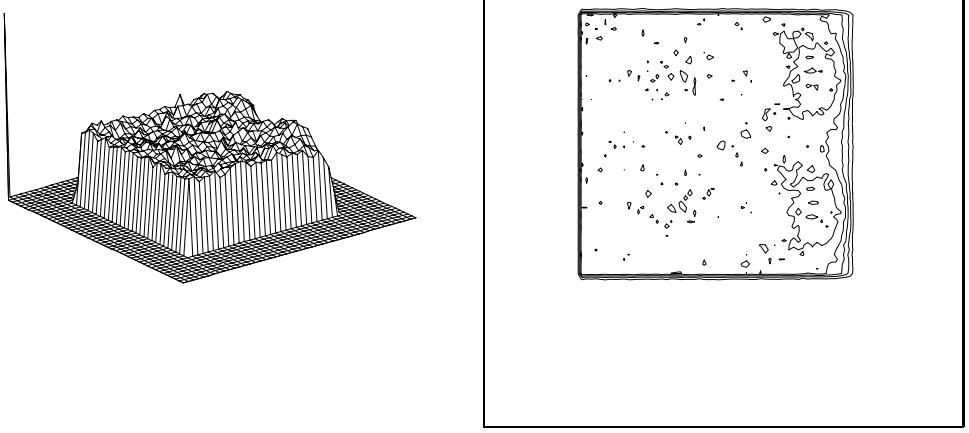

**Figure 5.2 Hash table equalization for the case of rigid transformations and point features uniformly distributed over the unit disc. Left: the expected distribution of remapped invariants. Right: several of the distribution's contours.**

An analysis similar to the one carried out for the case of the rigid transformations allows us to derive the following rehashing function

$$\boxed{\mathbf{h}(u,v) = \left(1 - \frac{3}{4\,(u^2+v^2)+3},\quad \text{atan2}(v,u)\right)}\,. \tag{5.3}$$

Use of this rehashing function allows the remapping of the computed invariants and results in the distribution shown in Figure 5.3. Again, the remapping is very efficient.

**Case of Affine Transformations.** We finally repeat the above analysis for the case of affine transformations where the point features are generated by a Gaussian process $\mathcal{N}(\mathbf{0}, \left(\begin{smallmatrix}\sigma & 0\\ 0 & \sigma\end{smallmatrix}\right))$. As was determined in Section 4.3, the distribution

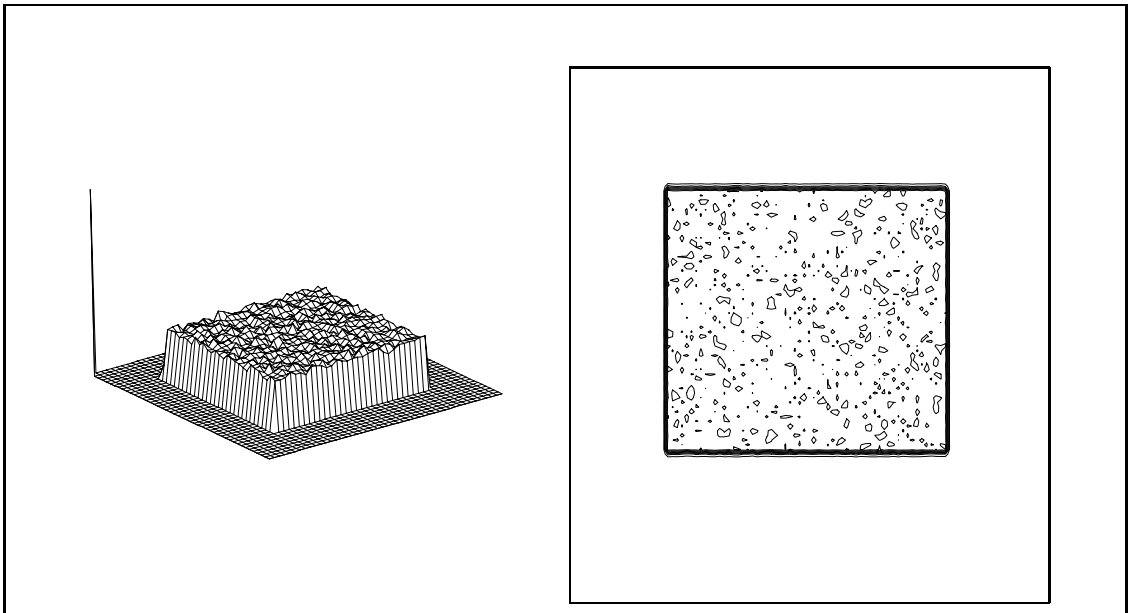

**Figure 5.3 Hash table equalization for the case of similarity transformations and point features generated by the Gaussian process $\mathcal{N}(\mathbf{0}, \left(\begin{smallmatrix} \sigma & 0 \\ 0 & \sigma \end{smallmatrix}\right))$. Left: the expected distribution of remapped invariants. Right: several of the distribution's contours.**

of indices in the space of invariants is given by

$$f(u,v) = \frac{2\sqrt{2}}{\pi} \frac{1}{(4u^2 + 4v^2 + 4uv + 8/3)^{\frac{3}{2}}}.$$

The appropriate rehashing function for this case can be shown to be

$$\mathbf{h}(u,v) = \left(1 - \frac{2\sqrt{2}}{\sqrt{3}\sqrt{(4u^2 + 4uv + 4v^2 + 8/3)}},\ \text{atan2}\left(u\sqrt{3} + v\sqrt{3}, u - v\right)\right). \quad (5.4)$$

In Figure 5.4, we show the result of hash table equalization, and several of the corresponding contours. As can be seen, the remapping is very efficient.

## 5.2 Symmetries and Foldings

In the previous section, we described the use of rehashing functions which results in a decrease of computational requirements. Additional savings in both compu-

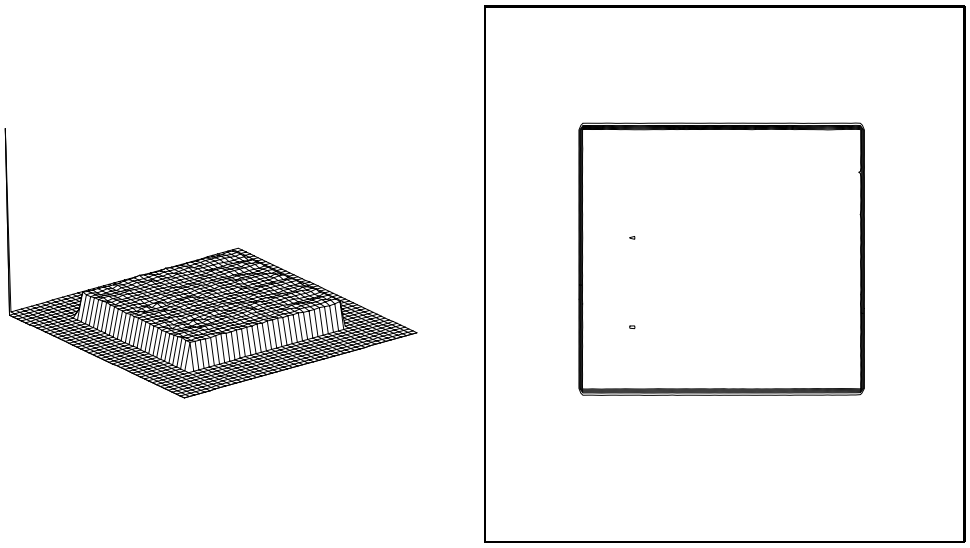

**Figure 5.4 Hash table equalization for the case of affine transformations and point features generated by the Gaussian process $\mathcal{N}(\mathbf{0}, \left(\begin{smallmatrix} \sigma & 0 \\ 0 & \sigma \end{smallmatrix}\right))$. Left: the expected distribution of remapped invariants. Right: several of the distribution's contours.**

tational and storage requirements are also possible. Certain symmetries exist in the storage pattern of entries in the hash table; these symmetries are independent of the use of rehashing functions given in section 5.1.

Let $\mathbf{p}_{\mu_1}$ and $\mathbf{p}_{\mu_2}$ be a basis pair comprising point features that belong to model $m$, and let $(u, v)$ be the coordinates of point $\mathbf{p}$ of $m$ in the coordinate system defined by $\mathbf{p}_{\mu_1}$, $\mathbf{p}_{\mu_2}$. Then, there will be an entry of the form $(m, (\mu_1, \mu_2))$ at location $(u, v)$ of the hash table (or at the rehashed position $\mathbf{h}(u, v)$). If the tuple $(\mathbf{p}_{\mu_2}, \mathbf{p}_{\mu_1})$ is used instead to form a basis pair, and thus a coordinate system, we observe that the coordinates of $\mathbf{p}$ will be $(-u, -v)$. I.e., at the location $(-u, -v)$ of the hash table there will be an entry of the form $(m, (\mu_2, \mu_1))$. This will hold true for both the rigid and similarity transformations. Due to the symmetry

of the rehashing functions (see Eqns. 5.1-5.3), the rehashed points $\mathbf{h}(u, v)$ and $\mathbf{h}(-u, -v)$ will still be related; in particular, they will have the same abscissa but they will be a distance $\pi$ apart in the polar coordinates. Figure 5.5 details the above observations for the special case of model $M_1$, and the basis tuples $(\mathbf{p}_4, \mathbf{p}_1)$, $(\mathbf{p}_1, \mathbf{p}_4)$.

A similar observation can be made for the case of the affine transformation as well. Let $\mathbf{p}$, $\mathbf{p}_{\mu_1}$ $\mathbf{p}_{\mu_2}$ and $\mathbf{p}_{\mu_3}$ be an arrangement of four points belonging to model $m$. There are a total of 4! distinct affine bases that one can form using points from this point set. Two such bases are $(\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_2}, \mathbf{p}_{\mu_3})$ and $(\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_3}, \mathbf{p}_{\mu_2})$. From Eqn. 2.3 we can see that the coordinates of $\mathbf{p}$ in the coordinate systems defined by the two (skewed) bases will be $(u, v)$ and $(v, u)$ respectively. In other words, the two hash entries corresponding to the two basis tuples will be symmetric with respect to the line that is at a 45 degree angle with the horizontal axis. At location $(u, v)$ of the hash table there will be an entry of the form $(m, (\mu_1, \mu_2, \mu_3))$, whereas at location $(v, u)$ there will be an entry of the form $(m, (\mu_1, \mu_3, \mu_2))$. As a result of the symmetry of the rehashing function in Eqn. 5.4, the rehashed points $\mathbf{h}(u, v)$ and $\mathbf{h}(v, u)$ will still be related; in particular, they will have the same abscissa, and opposite sign $\theta$ values. In Figure 5.6, we graphically depict the above observations for the case of model $M_1$, and the basis tuples $(\mathbf{p}_4, \mathbf{p}_5, \mathbf{p}_1)$, $(\mathbf{p}_4, \mathbf{p}_1, \mathbf{p}_5)$.

The result of these symmetries is that for every entry in a certain half of the hash table, there is an equivalent entry in the other half, with the only change that the basis tuple (or part of it) is reversed. Thus, we can dispose of half the hash table: during the recognition phase, when a hash occurs to the *missing* half of the table, the corresponding entry can be generated from the stored one, at the

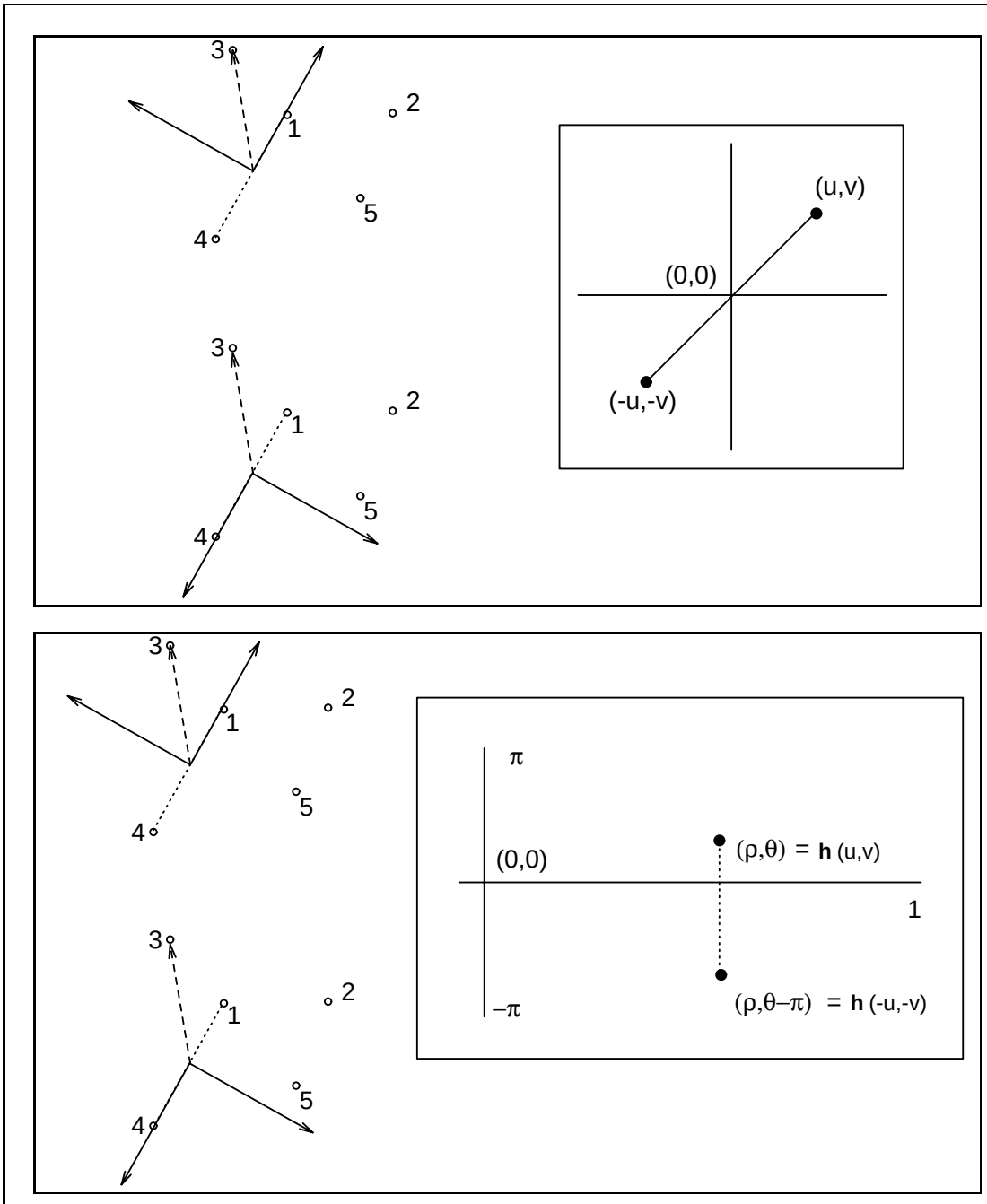


**Figure 5.5** Symmetries in the storage pattern of the hash entries. Top: if no rehashing is used, the hash entries are symmetric with respect to the center of the coordinate system of the space of invariants. Bottom: when rehashing is used, the rehashed entries have the same abscissa but are distance $\pi$ apart. These observations hold true for both the rigid and similarity transformations.

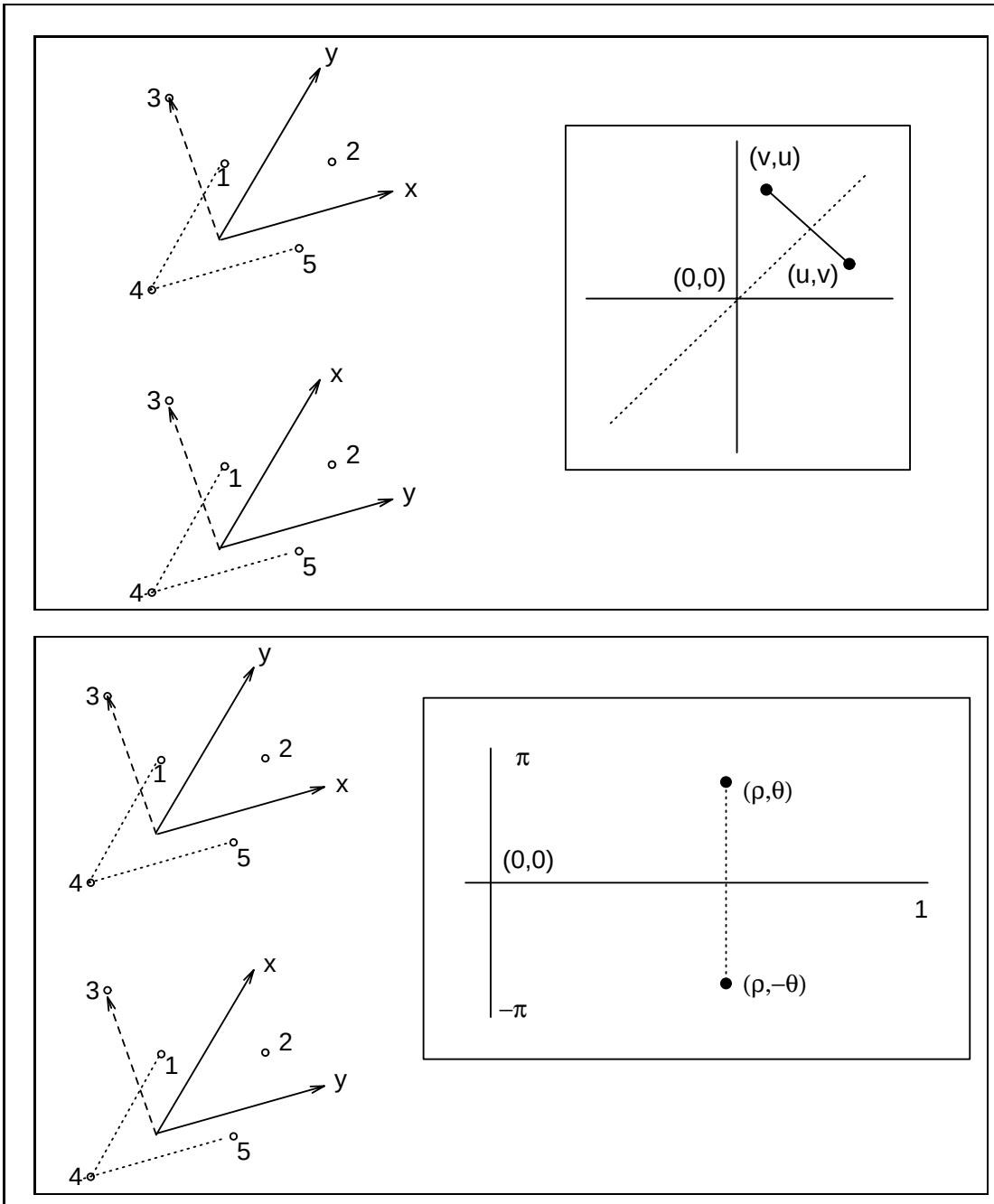


**Figure 5.6** Symmetries in the storage pattern of the hash entries under the affine transformation. Top: if no rehashing is used, the hash entries are symmetric with respect to the line that is at a 45 degree angle with the horizontal axis. Bottom: when rehashing is used, the rehashed entries have the same abscissa and $\theta$ values of opposite signs.

expense of minimal bookkeeping. Accordingly, only half the hash table will need to be stored, and thus the entry lists become, on the average, half as long, when spread among the existing set of processors. Consequently, a speedup of two, on the average, can be expected. It is interesting to note that this speedup occurs because of decreased storage requirements, and is not due to the doubling of the computational capacity.

Clearly, the computational savings accrued by the use of rehashing are irrelevant for the case of the broadcast algorithm. However, by making use of the symmetries in the storage pattern of the hash entries, the storage requirements for the broadcast algorithm can be reduced by a factor of two, leading to a corresponding reduction of the VPR value, and thus a speedup.

A further possibility is to perform a *folding* of the space of invariants. For example, when a hash occurs to the missing half-plane, rather than generating the entries from the list of entries in the associated position of the other half-plane, we can instead register a vote for the entire hash bin in the existing half-plane. In the case of similarity transformations, for example, this operation will in effect confuse entries of the form $(m, (\mu_1, \mu_2))$ with entries of the form $(m, (\mu_2, \mu_1))$: thus, the basis tuples are now basis *sets*, and $(\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_2})$ is the same as $(\mathbf{p}_{\mu_2}, \mathbf{p}_{\mu_1})$. Although this means that a particular $(model, basis)$ may receive more votes than it actually deserves, we have encountered no difficulties with this method.

## 5.3 Timing Results

In order to demonstrate the validity of our ideas, we tested the connectionist algorithm using synthetically generated models of 16 point features; the point features were generated by a Gaussian process. We only demonstrate the per-

formance enhancements of rehashing. The transformation class was the set of similarity transformations. During the building of the hash table, the rehashing function of Eqn. 5.3 was used. After generating 1024 such models, scenes were constructed of approximately 200 points, with a single model embedded in the scene, translated, rotated, and scaled. Noise was added to the scene points (through quantization round-off error).

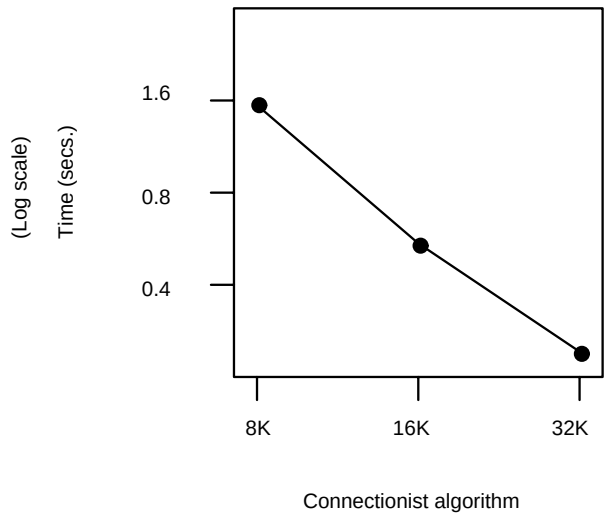


**Figure 5.7 Average time that the connectionist algorithm requires for a single basis probe, as a function of the number of processors in the Connection Machine. The database contains 1024 models of 16 points each; the points have been generated by a Gaussian process and the scenes contain 200 points. The allowed transformation is similarity and rehashing is used.**

The front end randomly selected a pair of scene points (a probe) as the basis to be used for possible recognition. For the connectionist algorithm, a probe takes 1.52 seconds on an $8K$-processor machine, dropping to 0.24 seconds on a $32K$-processor machine (see the plot in Figure 5.7), making use of rehashing but not using the symmetries. If the symmetries were used, then the time needed

for a probe would drop accordingly. Comparing these timing results to those of Figure 3.10, we can see how beneficial the performance enhancements are.

# Chapter 6

# Noise Modeling

In this chapter we introduce a new framework for model based object recognition in the context of geometric hashing. We incorporate additive Gaussian noise into our model and analytically determine its effect on the invariants that are computed for the case where the models are allowed to undergo similarity or affine transformations.

Usually, a number of drawbacks are associated with indexing techniques: namely, high sensitivity to sensor noise and to quantization of the space of the invariants (the "hash space"), poor index selectivity, and non-uniform accumulation of votes in the different bins. Solutions to the problem of non-uniform accumulation of votes, in the context of geometric hashing, were described for certain combinations of transformations and model point distributions in chapter 5. For noise tolerance, some preliminary results (see [36]) indicated that for a particular type of indexing, small amounts of sensor noise may lead to a performance degradation. Things are complicated by the fact that the index selectivity and the quantization coarseness of the space of invariants are interrelated. For example, if noise tolerance is important, the space of invariants is coarsely quantized, at the

expense of reduced index selectivity. If on the other hand, higher index selectivity is desired, the hash bins are made smaller, resulting in smaller noise tolerance. A common approach to the resolution of these competing demands is the use of higher-dimensional indices. Due to the fact that the minimum probability of error decreases as the dimensionality of the index increases [27], high-dimensional indexing results in more descriptive power (higher selectivity) and higher noise tolerance. However, the gains from this dimensionality increase cannot continue *ad infinitum*. Additionally, higher-dimensional indexing is not supported by physiological evidence [46,55].

The number of dimensions that are needed is typically task dependent and, either it is decided upon in advance [21,24], or it depends on the complexity of the objects stored in the database [89]. To our knowledge, no current system makes use of perceptual groupings as suggested by Lowe [68].

Recently, a number of researchers have attempted to build more robust object recognition systems by taking into consideration the effect of sensor noise. These efforts have met with moderate success. In particular, under the assumption of a bounded amount of noise for the sensing device, they derive either positional bounds [47,49,51], or bounds in the space of allowed transformations [18,48]. In all these systems, the databases contain only one or two models and the signal-to-noise ratio for the scene features (i.e. the ratio of points belonging to the model over the number of remaining scene points) is very small, typically one. Although bounds are appealing because of the tractability of the computations and the ease with which they can be applied, we conjecture that they will be of little or no help in the case of cluttered scenes or large databases; the reason is that they treat all the points of the bounded region equiprobably. An approach where the

sensing device is assumed to introduce Gaussian additive noise to the positions of the scene features should lead to improved results.

## 6.1 Performance in the Presence of Noise

In this section, we examine the performance of the geometric hashing approach in the presence of noise and discuss some suggestions that have been made in order to improve the performance.

We first examine the power of the geometric hashing method as a filtering method. In particular, we have designed and carried out a series of experiments that determine the percentage of model/basis combinations which for a given probe receive exactly $k$ votes, $k = 1, 2, 3, \ldots, n$; $n$ is the number of points in the model [79]. These experiments cover the cases of rigid and similarity transformations and were performed using large databases (512 models) and a large number (40) of test scenes. Similar experiments were reported in [60]; however, the databases used there were very small (no more than 20 objects), and the test scenes contained very little clutter (roughly as many clutter points as model points). In our experiments, there was only one model embedded in the scene. All of our database models consist of 16 point features. Our test scenes contain a total of 200 points (i.e. the signal-to-noise ratio for the scene features was almost 1/12).

Figure 6.1 graphically shows the results of some of the experiments, for the case where the database models are allowed to undergo similarity (i.e. rotation, translation and scaling) transformations. As can be seen from these graphs, the filtering power of the method is satisfactory: indeed, less than 2% of all possible

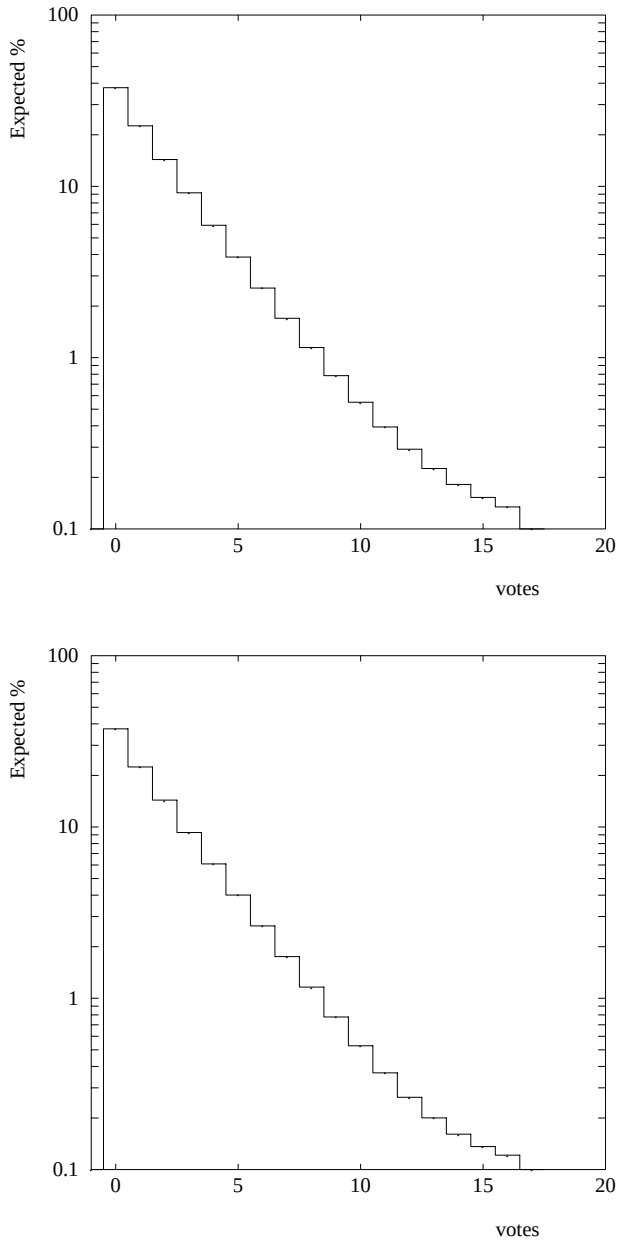


**Figure 6.1 Similarity Transforms: the expected percentage of model/basis combinations receiving exactly k votes. Top: the models' feature points are distributed according to a Gaussian of $\sigma$=1. Bottom: the models' feature points are distributed uniformly over the unit disc. In both cases, the database contained 512 models, each consisting of 16 points.**

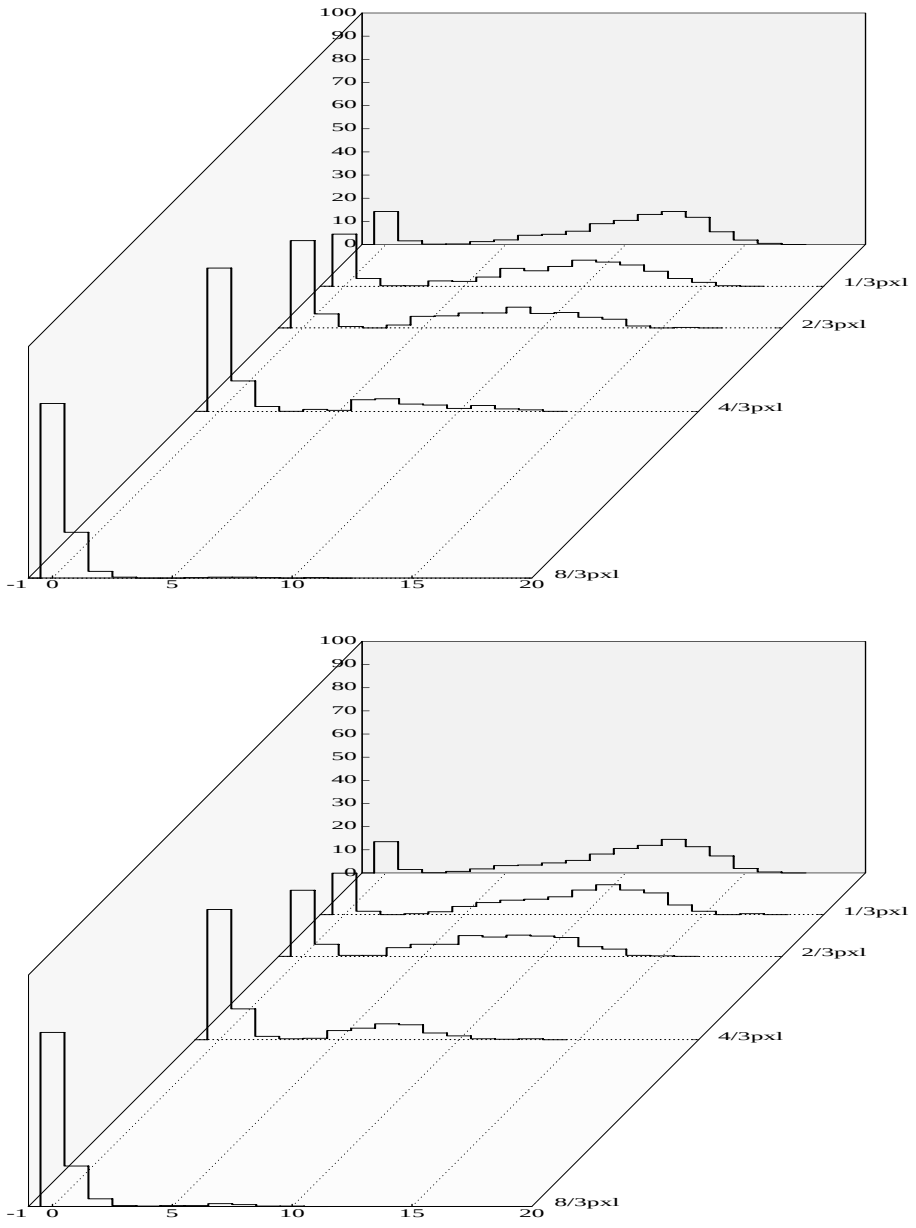


**Figure 6.2** Percentage of the embedded model's bases receiving k votes when used as probes, for different amounts of Gaussian noise. The models can only undergo similarity transformations. Top: the models points are distributed according to a Gaussian of $\sigma$=1. Bottom: the models points are distributed uniformly over the unit disc. In both cases, the database contained 512 models, each consisting of 16 points.

model/basis combinations receive more than 9 votes. Figure 6.2, on the other hand, shows the degradation of the method's performance as a function of the noise. Observe that little noise in the input suffices to render the model that is embedded in the scene practically undetectable. The reason is that the existence of noise in the input leads to positional errors, which in turn translate to errors in the invariants. If the error in the input is "small," the computed invariant, after proper quantization, will generate the same index as the noise-free case thus hashing into the correct hash table bin. Clearly, the semantics of "small" in this context directly depends on the coarseness of quantization of the space of invariants. Once this coarseness is decided, an associated degree of tolerance is implicitly built into the hash table. One can easily envision cases where a given hash table has insufficient power to discriminate among the stored models, for certain choices of features in the scene: the hash table is too coarsely quantized for these models; consequently, the hash table will yield too many candidate matches during the recognition phase and one must revert to a situation of testing and verifying many generated hypotheses.

Clearly, if a system is to be robust in a real setting, one should be able to cope with the problems caused by the positional uncertainty of the scene features.

In [60], it was suggested that during the selection process a region of the hash table (range of hash table bins) be accessed instead of a single bin. This region might have a rectangular shape and should be centered at the hash bin which is derived by the computed invariants. The same approach was suggested for both the similarity and the affine transformation cases.

Most geometric hashing systems to date have used a quantized space of invariants, so that entries fall in bins. Whenever a hash location is computed, then all

entries in the bin receive votes. Alternatively, we may assume that the space of invariants is quantized into tiny bins, and that a hash to a bin invokes votes for all entries in all bins in a circular region about the central bin. This is the approach used by Gavrila and Groen [34] and is implicit in the analysis of Huttenlocher and Grimson [36]. Specifically, this method approximates a weighted-voting function that assigns a unit vote for all entries located in a disk or rectangle centered at the hash location, and zero votes outside the disk or rectangle. Costa *et. al.* [25] have suggested, and we have used in our work, a weighted-voting function in the space of invariants that tapers down to zero in a region about the hash location, in a way that depends on the hash location and the basis selection in the scene. Thus if $(u, v)$ is the hash location, and $(u', v')$ is the location of a nearby entry, then the later entry will receive a weighted vote of $w(u, v, u', v')$. The function $w(\cdot, \cdot, \cdot, \cdot)$ may also depend on the scene basis that has been selected as a basis probe.

It is anticipated that voting for regions of the hash table instead of individual hash table bins will increase the degree of cross-talk among the different models. Indeed, a larger number of distinct model/basis combinations is expected to receive votes. Consequently, the verification step of the recognition phase will be more expensive. One might also observe an increased probability of false matches, although this is not likely an issue for small databases. As we will see in chapters 7 and 8, combining the idea of voting for regions with the idea of weighted voting [42] considerably improves the results. A first attempt at using weighted voting was described in [25]. The reported results were not very encouraging despite the fact that the database contained only one model.

Although the above ideas are reasonable, they do not capture the nature of the

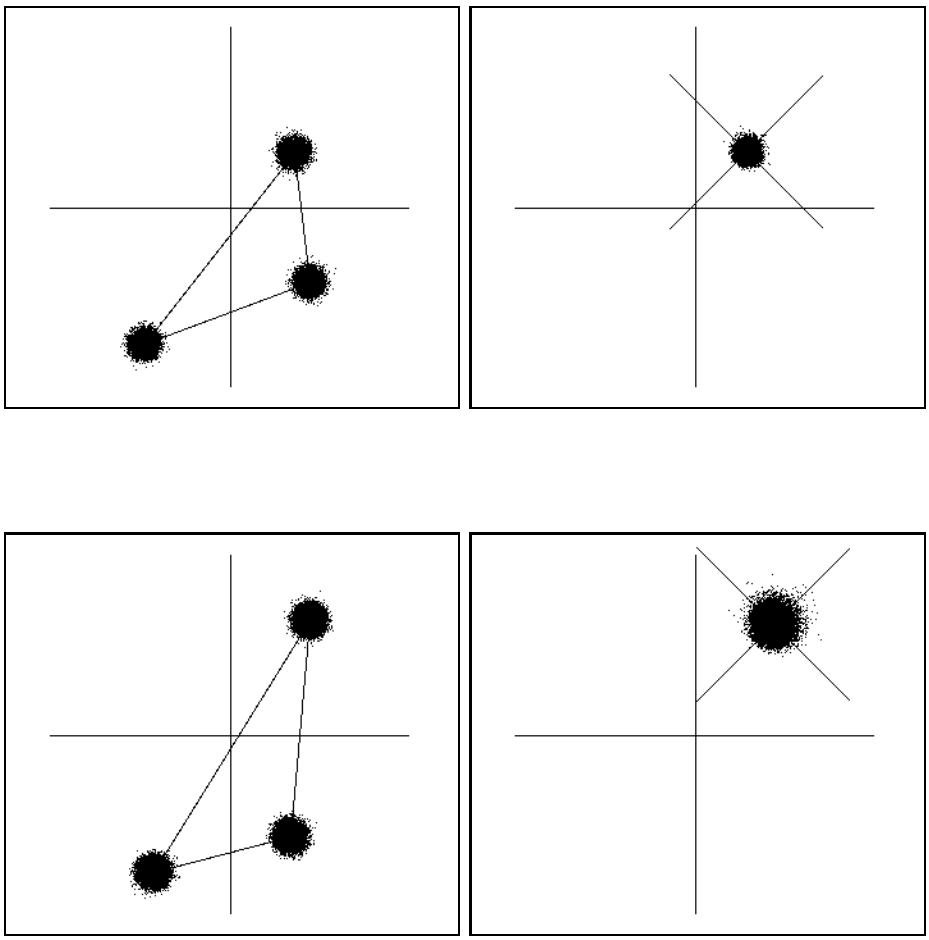

**Figure 6.3** Regions of the hash table that would need to be accessed in the case of Gaussian error in the positions of the point features. The models are allowed to undergo a similarity transformation. The left graph of each pair shows the feature space domain, whereas the right shows the space of invariants. For presentation purposes, the amount of Gaussian error was deliberately large.

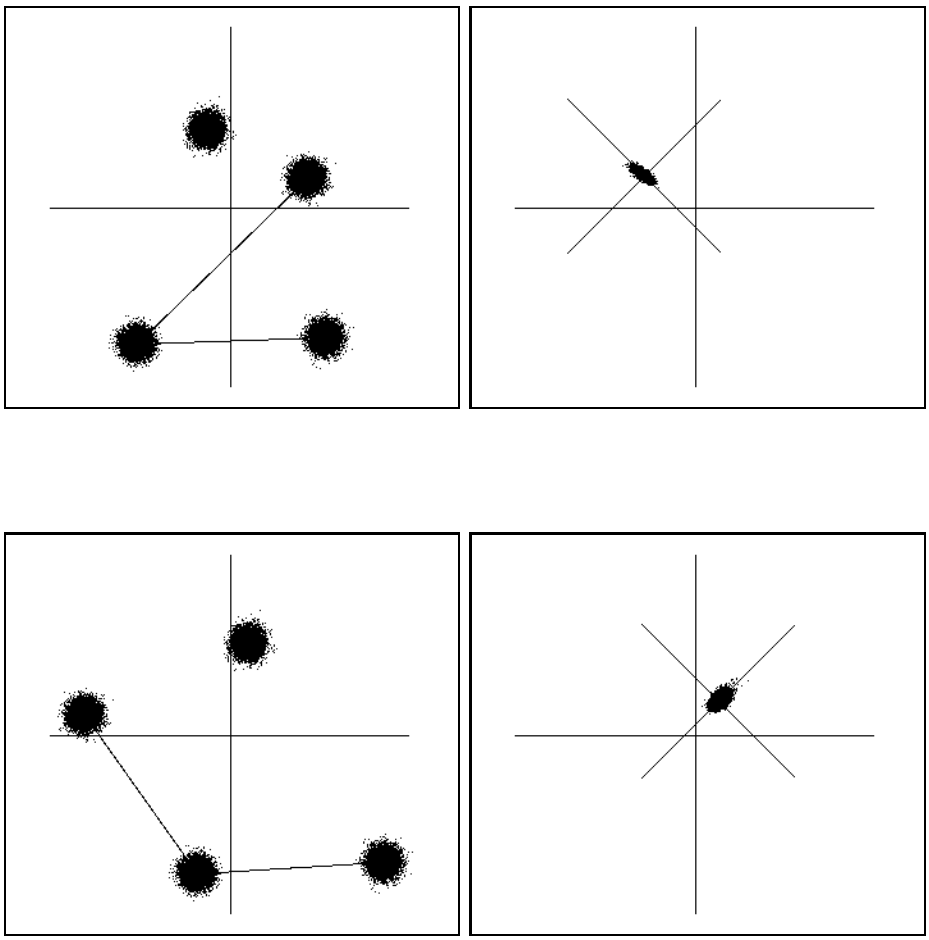

**Figure 6.4** Regions of the hash table that would need to be accessed in the case of Gaussian error in the positions of the point features. The transformation class is affine transformations. The left graph of each pair shows the feature space domain, whereas the right shows the space of invariants. For presentation purposes, the amount of Gaussian error was deliberately large.

problem. In particular, the size, shape and orientation of the regions that need to be accessed directly depend on the selected basis tuple, as well as on the computed hash locations. Figures 6.3 and 6.4 show this dependence for certain point configurations. As can be seen, the region variations are much more pronounced in the affine transformation case. These two figures provide a sound argument against the straightforward use of either Manhattan or Euclidean distances when determining the size of the regions that need to be accessed. Clearly, an *adaptive* scheme is needed, and the first step towards creating a working system that can perform satisfactorily in the presence of noise is the derivation of formulas quantifying the observed behavior. As we will show next, such formulas are easy to derive, and, furthermore, they are compatible with a Bayesian interpretation of geometric hashing.

## 6.2 Modeling Positional Noise

In this section we present a model for sensor noise and describe the effect of positional error on the values of the computed hash locations. We concentrate on the cases where the models are allowed to undergo similarity or affine transformations.

Traditionally, sensor noise has been modeled as additive Gaussian perturbations. The perturbations are assumed to be statistically independent and distributed according to a Gaussian distribution of standard deviation $\sigma$, centered at the "true" value of the variable. This is the noise model we will assume in our analysis of the dependence of the computed hash locations on noise in the input.

**Case: Similarity Transformation.** Let us consider a scene that contains $S$ feature points. Let $(x_i, y_i)$ be the "true" location of the $i$-th feature point in

the scene. Let also $(X_i, Y_i)$ be the continuous random variables denoting the coordinates of the $i$-th feature as these are measured by the sensing device (in this case a camera). The joint probability density function (pdf) of $X_i$ and $Y_i$ is then given by:

$$f(X_i, Y_i) = \frac{1}{2\pi\sigma^2} \exp(-\frac{(X_i - x_i)^2 + (Y_i - y_i)^2}{2\sigma^2}),\ i = 1, 2, \ldots, S$$

where we have assumed that the standard deviation value, $\sigma$, applies to both $X_i$ and $Y_i$, for all the values of $i$.

Let $(x_1, y_1)$ and $(x_2, y_2)$ define an *ordered* basis (and thus a coordinate frame $Oxy$). Let also $(x, y)$ be a third feature point whose coordinates $(u, v)$ in the frame $Oxy$ we wish to compute. Solving Eqn. 2.2 for $u$, $v$ yields

$$u = \frac{(\mathbf{p} - \mathbf{p}_0)(\mathbf{p}_2 - \mathbf{p}_1)^t}{\| \mathbf{p}_2 - \mathbf{p}_1 \|^2} \tag{6.1}$$

$$v = \frac{(\mathbf{p} - \mathbf{p}_0)(\mathbf{p}_2 - \mathbf{p}_1)^{\perp^t}}{\| (\mathbf{p}_2 - \mathbf{p}_1)^\perp \|^2} \tag{6.2}$$

or, assuming that $U$ and $V$ are the random variables denoting coordinates in the space of invariants,

$$U = \frac{(X - X_0, Y - Y_0)(X_2 - X_1, Y_2 - Y_1)^t}{\| (X_2 - X_1, Y_2 - Y_1)) \|^2} \tag{6.3}$$

$$V = \frac{(X - X_0, Y - Y_0)\,(X_2 - X_1, Y_2 - Y_1)^{\perp^t}}{\| (X_2 - X_1, Y_2 - Y_1)^\perp \|^2} \tag{6.4}$$

From probability theory, the joint pdf of $U$ and $V$ can be expressed as

$$f(U, V) = \int_{\mathbb{R}^4} f(X(U, V), Y(U, V)) f(X_1, Y_1) f(X_2, Y_2) \mid J \mid^{-1} dX_1 dX_2 dY_1 dY_2 \tag{6.5}$$

where $\mid J \mid^{-1} = \| (X_2 - X_1, Y_2 - Y_1) \|^2$. Since the pdf's $f(X, Y)$ $f(X_1, Y_1)$ and

$f(X_2, Y_2)$ are known, $f(U, V)$ can be computed from Eqn. 6.5 yielding:

$$\begin{aligned} f(U,V) \;=\; & \frac{1}{2\pi\sigma^2\left(4 \parallel (U,V) \parallel^2 +3\right)^3} \\ & \cdot \Big[ \; 2 \parallel \mathbf{p}_2 - \mathbf{p}_1 \parallel^2 \Big( \Big( 4\,(U,V)\,(u,v)^t + 3 \Big)^2 \\ & \quad + \Big( 4\,(U,V)\,(u,v)^{\perp^t} \Big)^2 \Big) + 24\sigma^2 \left(4 \parallel (U,V) \parallel^2 +3\right) \; \Big] \\ & \cdot \exp\left( - \frac{\parallel \mathbf{p}_2 - \mathbf{p}_1 \parallel^2}{\sigma^2} \cdot \frac{\parallel (U,V) - (u,v) \parallel^2}{4 \parallel (U,V) \parallel^2 +3} \right) \end{aligned} \quad . \quad (6.6)$$

In Appendix A, we give more details on the derivation of this formula.

Given a Gaussian perturbation of the three feature positions, Eqn. 6.6 describes the "spread" over the space of invariants of the computed coordinates. This formula substantiates the observations based on experiments that the shape and the size of the spread depend on the basis pair as well as on the value of the computed hash location.

The above formula is too complicated to be practical. For all practical purposes, the second and higher-order components of the expression in Eqn. 6.6 are not of any importance. It would thus be beneficial to derive a first order approximation for the spread over the space of invariants. To this end, we observe that the Eqns. 6.1 and 6.2 form the solution to the matrix equation:

$$\underbrace{\begin{pmatrix} x_2 - x_1 & -y_2 + y_1 \\ y_2 - y_1 & x_2 - x_1 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} u \\ v \end{pmatrix}}_{\mathbf{x}} = \underbrace{\begin{pmatrix} x - \frac{x_1+x_2}{2} \\ y - \frac{y_1+y_2}{2} \end{pmatrix}}_{\mathbf{b}} . \quad (6.7)$$

Let us now introduce Gaussian perturbation in the positions of the three points. Then, Eqn. 6.7 can be rewritten as follows:

$$(A + (\boldsymbol{\delta} A))\,\mathbf{x} = \mathbf{b} + (\boldsymbol{\delta}\mathbf{b}) \Rightarrow A\left(I + A^{-1}\,(\boldsymbol{\delta} A)\right)\mathbf{x} = \mathbf{b} + (\boldsymbol{\delta}\mathbf{b}) \Rightarrow$$

$$\mathbf{x} = \left(I + A^{-1}(\boldsymbol{\delta} A)\right)^{-1} A^{-1}(\mathbf{b} + (\boldsymbol{\delta}\mathbf{b})) \Rightarrow \mathbf{x} \approx \left(I - A^{-1}(\boldsymbol{\delta} A)\right) A^{-1}(\mathbf{b} + (\boldsymbol{\delta}\mathbf{b})) \Rightarrow$$

$$\mathbf{x} \approx A^{-1}\mathbf{b} + A^{-1}(\boldsymbol{\delta}\mathbf{b}) - A^{-1}\ (\boldsymbol{\delta} A)\ A^{-1}\mathbf{b} \tag{6.8}$$

where we have ignored second and higher order perturbation terms. We see that the Gaussian noise induces a perturbation $\boldsymbol{\delta}\mathbf{x}$ to the correct solution $\hat{\mathbf{x}} \equiv (u, v)$ that is equal to:

$$\boxed{\boldsymbol{\delta}\mathbf{x} = A^{-1}(\boldsymbol{\delta}\mathbf{b}) - A^{-1}(\boldsymbol{\delta} A)\ \hat{\mathbf{x}}}\ .$$

If $(\mathbb{X}, \mathbb{Y})$, $(\mathbb{X}_i, \mathbb{Y}_i)_{i=1,2}$, $(\mathbb{U}, \mathbb{V})$, are stochastic variables denoting the *perturbations* of $\mathbf{p}$, $\{\mathbf{p}_i\}_{i=1,2}$ and $(u, v)$ respectively, then the last equation can be rewritten as:

$$\begin{pmatrix} \mathbb{U} \\ \mathbb{V} \end{pmatrix} = \begin{pmatrix} -ku(\mathbb{X}_2 - \mathbb{X}_1) & +kv(\mathbb{Y}_2 - \mathbb{Y}_1) & -k/2(\mathbb{X}_1 + \mathbb{X}_2) & +k\mathbb{X} \\ -lu(\mathbb{Y}_2 - \mathbb{Y}_1) & -lv(\mathbb{X}_2 - \mathbb{X}_1) & -l/2(\mathbb{Y}_1 + \mathbb{Y}_2) & +l\mathbb{Y} \\ \\ -mu(\mathbb{X}_2 - \mathbb{X}_1) & +mv(\mathbb{Y}_2 - \mathbb{Y}_1) & -m/2(\mathbb{X}_1 + \mathbb{X}_2) & +m\mathbb{X} \\ -nu(\mathbb{Y}_2 - \mathbb{Y}_1) & -nv(\mathbb{X}_2 - \mathbb{X}_1) & -n/2(\mathbb{Y}_1 + \mathbb{Y}_2) & +n\mathbb{Y} \end{pmatrix}$$

where we have substituted $A^{-1}$ by $\left(\begin{smallmatrix} k & l \\ m & n \end{smallmatrix}\right)$ and $k, l, m, n \in \mathbb{R}$. Analogously to Eqn. 6.6, we have that the joint pdf, $f(\mathbb{U}, \mathbb{V})$, of $\mathbb{U}$ and $\mathbb{V}$ is equal to:

$$f(\mathbb{U}, \mathbb{V}) = \int_{\mathbb{R}^4} f(\mathbb{X}(\mathbb{U}, \mathbb{V}), \mathbb{Y}(\mathbb{U}, \mathbb{V})) f(\mathbb{X}_1, \mathbb{Y}_1) f(\mathbb{X}_2, \mathbb{Y}_2) \mid kn - ml \mid^{-1} d\mathbb{X}_1 d\mathbb{X}_2 d\mathbb{Y}_1 d\mathbb{Y}_2 .$$

Since $\mathbb{U}$, $\mathbb{V}$ are linear combinations of normally distributed stochastic variables, the joint distribution of $\mathbb{U}$, $\mathbb{V}$ will again be normal. Indeed, evaluation of the above integral yields for the probability density of the perturbation of $\hat{\mathbf{x}}$

$$\boxed{\begin{array}{c} f(\mathbb{U}, \mathbb{V}) = \dfrac{1}{2\pi\sqrt{\mid \Sigma_s \mid}} \exp\left(-\dfrac{1}{2}(\mathbb{U}, \mathbb{V})\, \Sigma_s^{-1}\, (\mathbb{U}, \mathbb{V})^t\right) \\ \\ \Sigma_s = \dfrac{(4 \parallel (u, v) \parallel^2 + 3)\, \sigma^2}{2 \parallel \mathbf{p}_2 - \mathbf{p}_1 \parallel^2} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array}}\ . \tag{6.9}$$

In other words, if we ignore second and higher-order terms, additive Gaussian positional error results in computed values for the similarity invariants that are also Gaussian distributed around their "true" value with covariance matrix $\Sigma_s$. A study of the last equation shows that it indeed incorporates the first order terms from Eqn. 6.6.

As can be seen from the expression for the *covariance* matrix $\Sigma_s$, the larger the separation of the two basis points the smaller the spread in the space of invariants; this is a long-honored observation. Eqn. 6.9 equation also introduces a new result. Indeed, for a given basis separation, the distance of the point whose coordinates we compute in the coordinate frame of the basis also affects the spread: the smaller this point's distance from the center of the coordinate frame, the smaller the spread in the space of invariants will be. The exact dependence is described by Eqn. 6.9. In chapter 4, we showed that the distribution of the entries over the space of invariants is non-uniform: hash indices near the center of the hash table are more frequent than indices further away. Given the result of Eqn. 6.9, we can see that there exists a trade-off between the indexing power of an invariant tuple and its sensitivity to noise. Although index values corresponding to relatively unpopulated regions of the space of invariants carry more information, they are very sensitive to noise. The opposite is also true: indices hashing near the peak of the hash table distribution (very populated area) are less informative but also more tolerant to noise. Recapitulating, we see that it is the entire triplet which determines the size of the spread and not only the basis pair. In other words, there is a distinct covariance matrix for every ordered triplet one can form using model points.

**Case: Affine Transformation.** The above analysis can be repeated for the case of the affine transformation. Let $(x_1, y_1)$, $(x_2, y_2)$, and $(x_3, y_3)$ define an ordered triplet and thus a coordinate frame $Oxy$. Let also $(x, y)$ be a third feature point whose coordinates $(u, v)$ in the frame $Oxy$ we wish to compute. We will consider the general case (Eqn. 4.5) where the center of the (skewed) coordinate system defined by the basis triplet is expressed as a function of the position vectors of the points comprising the basis. Eqn. 4.5 can be rewritten in matrix form as

$$\underbrace{\begin{pmatrix} x_2 - x_1 & x_3 - y_1 \\ y_2 - y_1 & y_2 - y_1 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} u \\ v \end{pmatrix}}_{\mathbf{x}} = \underbrace{\begin{pmatrix} x - \alpha x_1 - \beta x_2 - \gamma x_3 \\ y - \alpha y_1 - \beta y_2 - \gamma y_3 \end{pmatrix}}_{\mathbf{b}}. \quad (6.10)$$

Introduction of Gaussian perturbation in the positions of all four points, will result in a perturbation of the correct solution $\hat{\mathbf{x}} \equiv (u, v)$ by

$$\boxed{\boldsymbol{\delta}\mathbf{x} = A^{-1}(\boldsymbol{\delta}\mathbf{b}) - A^{-1}(\boldsymbol{\delta}A)\ \hat{\mathbf{x}}}.$$

Again the second and higher-order perturbation terms have been ignored. If $(\mathbb{X}, \mathbb{Y})$, $(\mathbb{X}_i, \mathbb{Y}_i)_{i=1,2,3}$, $(\mathbb{U}, \mathbb{V})$, are stochastic variables denoting the *perturbations* of $\mathbf{p}$, $\{\mathbf{p}_i\}_{i=1,2,3}$ and $(u, v)$ respectively, then Eqn. 6.10 can be rewritten as:

$$\begin{pmatrix} \mathbb{U} \\ \mathbb{V} \end{pmatrix} = \begin{pmatrix} -ku(\mathbb{X}_2 - \mathbb{X}_1) & -kv(\mathbb{X}_3 - \mathbb{X}_1) & -k(\alpha\mathbb{X}_1 + \beta\mathbb{X}_2 + \gamma\mathbb{X}_3) & +k\mathbb{X} \\ -lu(\mathbb{Y}_2 - \mathbb{Y}_1) & -lv(\mathbb{Y}_3 - \mathbb{Y}_1) & -l(\alpha\mathbb{Y}_1 + \beta\mathbb{Y}_2 + \gamma\mathbb{Y}_3) & +l\mathbb{Y} \\ -mu(\mathbb{X}_2 - \mathbb{X}_1) & -mv(\mathbb{X}_3 - \mathbb{X}_1) & -m(\alpha\mathbb{X}_1 + \beta\mathbb{X}_2 + \gamma\mathbb{X}_3) & +m\mathbb{X} \\ -nu(\mathbb{Y}_2 - \mathbb{Y}_1) & -nv(\mathbb{Y}_3 - \mathbb{Y}_1) & -n(\alpha\mathbb{Y}_1 + \beta\mathbb{Y}_2 + \gamma\mathbb{Y}_3) & +n\mathbb{Y} \end{pmatrix}$$

where we have substituted $A^{-1}$ by $\left(\begin{smallmatrix} k & l \\ m & n \end{smallmatrix}\right)$ and $k, l, m, n \in \mathbb{R}$. From standard

probability theory, the joint probability density, $f(U,V)$, of $U$ and $V$ is given by:

$$f(\mathbb{U},\mathbb{V}) = \int_{\mathbb{R}^6} f(\mathbb{X}(\mathbb{U},\mathbb{V}),\mathbb{Y}(\mathbb{U},\mathbb{V}))\; f(\mathbb{X}_1,\mathbb{Y}_1)\; f(\mathbb{X}_2,\mathbb{Y}_2)\; f(\mathbb{X}_3,\mathbb{Y}_3) \\ \mid kn - ml \mid^{-1} d\mathbb{X}_1 d\mathbb{X}_2 d\mathbb{X}_3 d\mathbb{Y}_1 d\mathbb{Y}_2 d\mathbb{Y}_3 \,. \tag{6.11}$$

Both $\mathbb{U}$, and $\mathbb{V}$ are linear combinations of normally distributed stochastic variables. Thus, the joint distribution of $\mathbb{U}$, $\mathbb{V}$ will again be normal. Evaluation of the above integral yields for the probability density of the perturbation of $\hat{\mathbf{x}}$

$$\boxed{\begin{aligned} f(\mathbb{U},\mathbb{V}) &= \frac{1}{2\pi\sqrt{\mid \Sigma_a \mid}} \exp\left(-\tfrac{1}{2}(\mathbb{U},\mathbb{V})\Sigma_a^{-1}(\mathbb{U},\mathbb{V})^t\right) \\ \Sigma_a &= \frac{\left(4\left(u^2+v^2+uv+(\beta-\alpha)\,u+(\gamma-\alpha)\,v\right)+2\left(\alpha^2+\beta^2+\gamma^2+1\right)\right)\sigma^2}{2\;\mid (\mathbf{p}_2-\mathbf{p}_1)\,(\mathbf{p}_3-\mathbf{p}_1)^{\perp^t}\mid^2} \\ &\quad \cdot \begin{pmatrix} \parallel (\mathbf{p}_3-\mathbf{p}_1)\parallel^2 & -(\mathbf{p}_2-\mathbf{p}_1)\,(\mathbf{p}_3-\mathbf{p}_1)^t \\ -(\mathbf{p}_2-\mathbf{p}_1)\,(\mathbf{p}_3-\mathbf{p}_1)^t & \parallel (\mathbf{p}_2-\mathbf{p}_1)\parallel^2 \end{pmatrix} \end{aligned}} \,. \tag{6.12}$$

In other words, if we ignore second and higher order terms, additive Gaussian positional error results in computed values for the affine invariants that are also Gaussian distributed around their "true" value with covariance matrix $\Sigma_a$.

It is clear from expression 6.12, that the spread around the noise-free solution will be small for small values of the positional error in the input, small values of the computed invariants, and long bases. Also, the smaller the skewness of the coordinate frame that the three model points define, the smaller the spread will be. In other words, there is a distinct covariance matrix for every ordered quadruplet that is formed using model points. For the special case where $\alpha = \beta = \gamma = 1/3$, the covariance matrix $\Sigma_a$ becomes

$$\boxed{\begin{aligned} \Sigma_a &= \frac{\left(4\left(u^2+v^2+uv\right)+8/3\right)\sigma^2}{2\;\mid (\mathbf{p}_2-\mathbf{p}_1)\,(\mathbf{p}_3-\mathbf{p}_1)^{\perp^t}\mid^2} \\ &\quad \cdot \begin{pmatrix} \parallel (\mathbf{p}_3-\mathbf{p}_1)\parallel^2 & -(\mathbf{p}_2-\mathbf{p}_1)\,(\mathbf{p}_3-\mathbf{p}_1)^t \\ -(\mathbf{p}_2-\mathbf{p}_1)\,(\mathbf{p}_3-\mathbf{p}_1)^t & \parallel (\mathbf{p}_2-\mathbf{p}_1)\parallel^2 \end{pmatrix} \end{aligned}} \,.$$

# Chapter 7

# Bayesian Interpretation

In this chapter we provide an interpretation of geometric hashing that shows that the algorithm is equivalent to a Bayesian maximum likelihood object recognition method. The hypotheses span the discrete collection of models and the discrete pairings of image features to model features.

The equivalence is precise only in the case where an adaptive weighted voting scheme is used to accumulate evidence for model/basis combinations in the geometric hashing framework. We provide formulas for the weighting functions in the case of similarity and affine-invariant recognition of point patterns and under the assumption of Gaussian positional error in the image points. Finally we discuss how the weighted-voting scheme and maximum-likelihood hypothesis selection reduces false alarms.

We begin the chapter by providing an *abstract* presentation of the geometric hashing method reformulating the more specific version given in section 2.2.1.

## 7.1 Abstract Formulation of Geometric Hashing

As we have already mentioned, geometric hashing operates on image *features* in order to locate *objects*. Image features are generated as the output of the feature extraction stage, and typically are local measurements which depend on the grey levels of an image region. Each feature typically results in a vector of measurements. Minimally, a feature carries only positional information, in which case it is a two-vector giving the coordinates of a location. However, there may be other measurements (attributes) associated with a feature, such as an angle measurement, a gray level value, a direction (for example, of a line), etc.

Henceforth, we will assume that the image features are elements of $\mathbb{R}^p$, and that $c$ image features are needed to determine a basis. Nominally, $p = 2$, i.e. the features are points. The case $c = 2$ corresponds to rigid and similarity invariance whereas $c = 3$ corresponds to affine invariance. Depending on the feature type and invariance class, other combinations of $p$ and $c$ are also possible. We will further assume that the class of possible transformations, under which we desire invariant recognition, is given by $F$. The members of $F$ act on image features.

The database contains $M$ models: $1, 2, \ldots, M$. Each model $m$ is a collection of $n$ features, $\mathcal{F}_m = \{\mathbf{f}_{m,k}\}_{k=1}^{n}$. For simplicity, we are assuming that the number of features is the same across the models.

For recognition, we are given an image $\mathcal{S}$, containing a total of $S$ features, $\mathcal{S} = \{\mathbf{p}_l\}_{l=1}^{S}$. In the terminology of Flynn [32] we seek a collection of interpretations. An *interpretation* is a tuple

$$[m, [\mathbf{f}_{m,j_1}, \mathbf{p}_{k_1}], [\mathbf{f}_{m,j_2}, \mathbf{p}_{k_2}], \ldots, [\mathbf{f}_{m,j_r}, \mathbf{p}_{k_r}]]$$

where $m$ gives the index number of a model ($1 \leq m \leq M$), and the remaining $r$

pairs establish approximate correspondences between a subset of model features of $\mathcal{F}_m$ with image features. Typically an interpretation involves more than $c$ matches (i.e. $r \geq c$) so that an interpretation carries an implicit transformation that approximately matches a subset of features in a model to a subset of image features and the pairings are all distinct. An interpretation with $r = c$ distinct pairs is a *candidate matching* (or simply a matching) which can be verified or rejected according to whether the matching can be extended to a viable interpretation with the number of matchings substantially greater than $c$. Note that *a priori* a recognition system must search over all possible matchings:

$$[m, [\mathbf{f}_{m,j_1}, \mathbf{p}_{k_1}], [\mathbf{f}_{m,j_2}, \mathbf{p}_{k_2}], \ldots, [\mathbf{f}_{m,j_c}, \mathbf{p}_{k_c}]]$$

to determine whether matchings can be extended to interpretations. Clearly since there are $M$ models $n$ features in each model and $S$ image features there are $\mathcal{O}(Mn^c S^c)$ candidate matchings.

**Hash Functions.** In the geometric hashing model we use a hash function that maps groups of $N = c+d$ features to $\mathbb{R}^e$. Here $d$ is the number of "extra features" and $e$ is the dimensionality of the space of invariants. Again nominally when we use point features we have $d = 1$ i.e. the hash function is defined on groups that consist of one more point than the basis set and $e = 2$ meaning that the hash function maps into the Cartesian space. But more generally the hash function uses collections of features to map into a higher dimensional space. The essential point about the hash function is that it is $F$-invariant. Thus if $\mathcal{T} \in F$ is a transformation and $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_N$ is a collection of $N$ features then

$$\mathbf{h}(\mathcal{T}\mathbf{p}_1, \mathcal{T}\mathbf{p}_2, \ldots, \mathcal{T}\mathbf{p}_N) = \mathbf{h}(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_N) \ .$$

Further, in order to enable recognition in the presence of noise, $\mathbf{h}$ must be a continuous function. This is in complete distinction with normal hashing methods for database search, where the hashing function typically randomizes the keys as much as possible. As such, hashing is not necessarily the most appropriate name for the technique using a continuous function $\mathbf{h}$; perhaps it should instead be called "geometric indexing."

An additional desirable property of the hash function is that the fibers coincide with equivalence classes of the transformation group. Thus if two collections of points have the same hash location:

$$\mathbf{h}(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_N) = \mathbf{h}(\mathbf{p}'_1, \mathbf{p}'_2, \ldots, \mathbf{p}'_N),$$

then there exists a transformation $\mathcal{T} \in F$ such that

$$(\mathcal{T}\mathbf{p}_1, \mathcal{T}\mathbf{p}_2, \ldots, \mathcal{T}\mathbf{p}_N) = (\mathbf{p}'_1, \mathbf{p}'_2, \ldots, \mathbf{p}'_N).$$

We will say that a hash function that satisfies this property is *F-specific*, as in, for example, *affine-transformation-specific*. An $F$-specific hash function is a bijection on equivalence classes of $N$-tuples of features modulo transformations from the class $F$. Systems are likely to be possible with hash functions that are not transformation-class-specific.

We can observe why the hash function must operate on $N$ features with $N > c$. If we define a hash function on a basis set or sub-basis set, then using a transformation, the hash value must be the same independent of the configuration of the features. That is, for any other configuration, we could map the initial basis set (or sub-basis set) to the given configuration, and the hash value should remain the same. A constant hash function is of little use.

On the other hand, in practice, features are not arbitrarily modifiable by the transformations. For example, if there are multiple feature types, then the type of the feature is usually not changed by a transformation. In this case, collections of features, even in a basis set, contain certain inherent information that is invariant with respect to the transformation class, and thus in this case, a hash function may be defined that takes advantage of this information. Our treatment here is more elementary and abstract, thus we assume that any collection of $c$ or fewer features may be mapped by a transformation to any configuration of the same number of features. Accordingly, a hash function on $c$ or fewer features will be constant.

However, a hash function defined on $N = c + d$ features will map into a Euclidean space, and can exhibit up to $d \cdot p$ degrees of freedom, where we recall that $p$ is the dimensionality of the features. Thus, the space of invariants can effectively span a $(d \cdot p)$-manifold, and is most reasonably parameterized by Euclidean space. Consequently, the dimensionality $e$ of the hash space will always be less than the number of extra degrees of freedom in the arguments to the hash function, i.e. $e \leq d \cdot p$. Thus if the hash function is defined on groups of $2D$ points consisting of a basis set plus one point, then the maximum effective dimensionality of the space of invariants will be two. However, if the hash function is defined on $s$ a basis sets plus two points, then we can have a four-dimensional space of invariants.

**Indexing.** Hash locations are used to index to model/basis combinations. Indexing is based on a two-phase process. In the first phase, the models are processed off-line, and the hash table data structure is built. During the preprocessing phase, hash values are computed for combinations of $N$ features from models. Ide-

ally for every set $\mathcal{F}_m$ for every combination of $N$ features $\{\mathbf{f}_{m,1}, \mathbf{f}_{m,2}, \ldots, \mathbf{f}_{m,N}\} \subset \mathcal{F}_m$ and for every permutation of these $N$ features $\mathbf{h}(\mathbf{f}_{m,1}, \mathbf{f}_{m,2}, \ldots, \mathbf{f}_{m,N})$ is computed. In practice certain permutations might be omitted and some combinations might be collapsed because of symmetries in the hash function. Normally at least $c$ of the model features will be distinct but there is no general reason for prohibiting repetitions. For each computed hash location in $\mathbb{R}^e$ we define an *entry* in the space of invariants. An entry consists of a tagged point in $\mathbb{R}^e$ such that given a point $(u, v)$ in the space of invariants a selection algorithm can quickly locate nearby entries in the hash space and access the tagged information on each such entry. One might reasonably use a $k-D$ tree representation for the set of all entries [74].

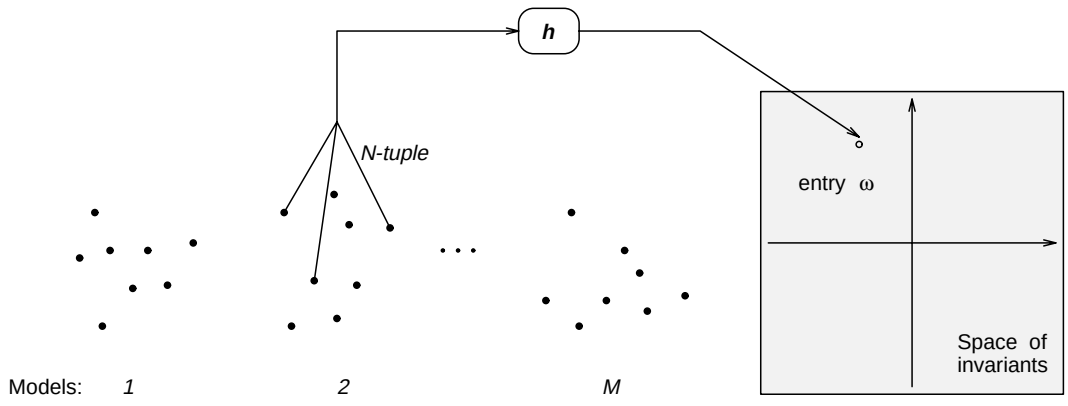


**Figure 7.1 The preprocessing phase: for each model and for every N-tuple of points in the model, a hash location is computed, and an entry is recorded in the space of invariants at that location. The entry is tagged with the information concerning the model identity and model features that were used to compute the position.**

What information is tagged at an entry? As a minimum the entry records its location in the space of invariants and the identity of the model that was used to

create the entry (i.e. model $m$). Typically the entry will also contain information allowing one to deduce the tuple of features from model $m$ $(\mathbf{f}_{m,j_1}, \mathbf{f}_{m,j_2}, \ldots, \mathbf{f}_{m,j_N})$ that was used to compute the location. Clearly it is sufficient to record the indices $(j_1, j_2, \ldots, j_N)$. Further information might be recorded with the entry. For example we could attach an error model giving the covariance of the expected distribution of the entry in the space of invariants as instances of the model in a typical image are subjected to noise. In Flynn's terminology the tagged information at an entry is called a *proto-hypothesis*.[1] A sketch of the preprocessing phase is depicted in Figure 7.1. More formally we define a hash entry as follows:

> **Definition:** An *entry* $\omega$ is a tuple of a model index and indices of an $N$-tuple of model points: $\omega = [m, [j_1, j_2, \ldots, j_N]]$ with a position denoted as $\boldsymbol{\zeta}(\omega) = \mathbf{h}(\mathbf{f}_{m,j_1}, \mathbf{f}_{m,j_2}, \ldots, \mathbf{f}_{m,j_N})$ and a tag $\boldsymbol{\tau}(\omega)$ which is typically the information $[m, [j_1, j_2, \ldots, j_c]]$ but might include additional information. □

The second phase is the **recognition** phase (on-line). We are presented with an image and $S$ features $\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_S\}$ are extracted. We then initiate a search in the image. Ultimately subcollections of $N$ features will be used to compute hash locations $\mathbf{h}(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_N)$. Once again repetitions of image features in the $N$-tuple might be allowed. For each such hash location we locate all nearby entries in the hash table and based on the information in each entry we generate one or more votes (see Figure 7.2). Each nearby entry generates a candidate interpretation. Specifically if the entry $\omega = [m, [j_1, j_2, \ldots, j_N]]$ is tagged

[1] Flynn [32] calls a quantized bin in space of invariants an *entry*. Obviously this use of the term clashes with our use of *hash entry* which refers to a tagged point in the space of invariants. Thus Flynn's entry is our hash bin and his proto-hypothesis is our hash entry.

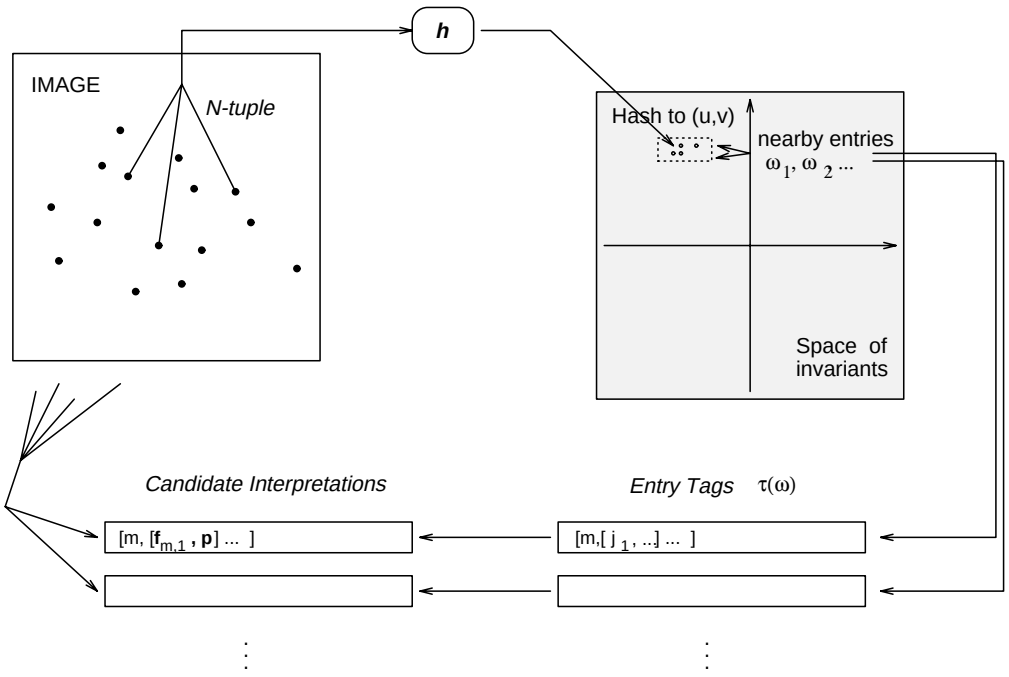


**Figure 7.2 The recognition phase and voting process of Bayesian geometric : hashing using N-tuples of image features, locations in the space of invariants are computed, and nearby entries are accessed. Each entry is tagged with a model number and a set of model features, which can be paired with the image features used to compute the hash location to form a candidate interpretation. Interpretations are then given weighted votes.**

with the information '(model $m$)' and model points $(\mathbf{f}_{m,j_1}, \mathbf{f}_{m,j_2}, \ldots, \mathbf{f}_{m,j_N})$ then the hash $\mathbf{h}(\mathbf{p}_{i_1}, \mathbf{p}_{i_2}, \ldots, \mathbf{p}_{i_N})$ to a nearby location generates a candidate interpretation:

$$[m, [\mathbf{f}_{m,j_1}, \mathbf{p}_{k_1}], [\mathbf{f}_{m,j_2}, \mathbf{p}_{k_2}], \ldots, [\mathbf{f}_{m,j_N}, \mathbf{p}_{k_N}]] \tag{7.1}$$

In certain cases (for example if the hash function is known to have certain symmetry properties) other interpretations might also be generated involving the same sets of features. Votes are generated for interpretations or perhaps sub-interpretations meaning a portion of the interpretation. For example we

might generate votes for model numbers (i.e. the index $m$ for the given entry) and then consider only models that receive sufficient votes in a subsequent search process. Alternatively we may generate votes for matches of the form $[m, [\mathbf{f}_{m,j_1}, \mathbf{p}_{k_1}], [\mathbf{f}_{m,j_2}, \mathbf{p}_{k_2}], \ldots, [\mathbf{f}_{m,j_c}, \mathbf{p}_{k_c}]]$ and later verify candidate matches that receive sufficient votes. In between it is possible to generate votes for submatchings together optionally with votes in Hough-space for transformation parameter values. However our practice is to generate votes for full interpretations of the form of Eqn. 7.1. However if two distinct collections of $N$ image points vote for the same entry then the two resulting interpretations are incompatible and arbitration must be performed. The most convenient time to perform the arbitration is when the hash takes place since both candidate interpretations may be "stored" at the hash table entry.

A single vote for a single interpretation may not in itself constitute much evidence for the existence of a model. Two interpretations are compatible however if they agree on pairs of model-feature and image-feature pairs and do not disagree on other features. Accordingly after collecting votes for multiple interpretations we will want to collapse the votes to discover multiple support for common sub-interpretations. Sub-interpretations with $c$ pairs of distinct features (i.e. matchings) are particularly desirable since they indicate a unique matching between a model and an image object together with the approximate transformation.

**Search Strategies.** Given an image hash values and votes are generated by $N$-tuples of image points. For an image with $S$ points there are $\mathcal{O}\left(S^N\right)$ such tuples. Each tuple can be used to locate a hash value and thus nearby entries

in the hash table. Each such entry can generate candidate interpretations which can involve up to $N$ feature pairs each.

We wish to structure the search over the $\mathcal{O}\left(S^N\right)$ tuples so as to quickly extract as many valid interpretations as possible. Note that the search space that occurs without the use of hashing is of size $\mathcal{O}(Mn^cS^c)$ so that hashing is more favorable when $M$ is large or in general when $S^d = o(Mn^c)$. Of course the trade-off depends on the effectiveness of the hash function and the typical number of interpretations that are generated during the search.

The best search strategy will depend on the application and will also be considerably influenced by the possible use of multiple feature types. However the following strategy is a generic one that structures the search and takes advantage of the possibility of "grouping" features in the image.

We iteratively chose basis sets of $c$ distinct features in the image. There are $\mathcal{O}(S^c)$ such sets. For each basis set we perform what we call a *probe*. The basis sets may be chosen at random or may make use of grouping or other strategies. It is desired that the $c$ features chosen in the image all belong to a single object. On a parallel machine multiple probes may be performed concurrently. In section 8.2 we describe a randomized algorithm that allows us to discard a probe after only a fraction of the image features have been considered.

Entries will obtain weighted votes during the probe; at the start of the probe all weights are zero. For each probe we append to the chosen basis set $\mathcal{B}$ collections of $d$-tuples of image features. For technical reasons we prefer to allow repetitions among the features in the $d$-tuple although some applications may ban such repetitions. In any case the $d$-tuples will be formed of image features that do not include any of the basis set. Using the tuple formed from $\mathcal{B}$ and a

$d$-tuple the hash function is applied to the resulting $N$-tuple to compute a hash location. Nearby entries receive weighted votes. If an entry has already received a non-zero vote during this probe then a collision occurs since distinct tuples of image features are competing with the same $N$-tuple of model points. We accept the vote with the *closer*[2] hash location which typically means the maximum vote. After all (or many) $d$-tuples of image features are processed each entry $\omega$ will have a weight $z(\omega)$ and we are ready to accumulate votes for matchings. An entry

$$\omega = [m, [j_1, j_2, \ldots, j_c, j_{c+1}, \ldots, j_N]]$$

adds its vote to a model/basis accumulator indexed by $[m, [j_1, j_2, \ldots, j_c]]$. This accumulator implicitly represents the interpretation that the chosen basis set $\mathcal{B}$ matches with the $c$ model features used as a basis in the computation of the corresponding entry. Note that the index of the accumulator to which the votes are collapsed is the same as the data that is usually included in the tag $\boldsymbol{\tau}(\omega)$. We may write:

$$Z(m, [j_1, j_2, \ldots, j_c], \mathcal{B}) = \sum_{j_{c+1}} \cdots \sum_{j_N} z([m, [j_1, j_2, \ldots, j_c, j_{c+1}, \ldots, j_N]]) \ .$$

If for a given probe no model/basis combination receives a sufficient number of votes then we deem that there is no valid interpretation and continue with another probe (a different basis set) in the image. If instead there are model/basis combinations with a substantial weighted vote then each such implicit matching should be verified.

If all $\mathcal{O}(S^c)$ probes are performed and $\mathcal{O}\left(S^d\right)$ different tuples are used in each probe then a complete search of $\mathcal{O}\left(S^N\right)$ will be invoked. However by

[2] A Mahalanobis distance metric is used here.

organizing the search over basis sets in the image votes may be accumulated for interpretations that involve exactly $c$ pairs of features and basis sets in the image that are likely to lead to recognition may be favored over completely random selections. Further an object is typically recognized once any combination of $c$ features on the object is selected as a basis. Since there is a $\binom{n}{c}$-fold redundancy in the representation of the objects there is a concomitant expected speed-up.

## 7.2 Updating Formulas and Conditional Independence

Let $\mathcal{H}$ be a hypothesis such as "object $\mathbb{O}$ is present in a designated region of the image." Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be two pieces of evidence that have impact on the probability of $\mathcal{H}$. In what follows we assume that $\mathcal{E}_1$ and $\mathcal{E}_2$ are boolean events although extensions to predicates involving continuous-valued measurements are straightforward. Our interest is to compute

$$\Pr(\mathcal{H} \mid \mathcal{E}_1, \mathcal{E}_2).$$

Let us assume that we know in advance the values of $\Pr(\mathcal{H} \mid \mathcal{E}_1)$ $\Pr(\mathcal{H} \mid \mathcal{E}_2)$ and the prior probability $\Pr(\mathcal{H})$.

In general there is nothing that can be said. The functional dependence of $\Pr(\mathcal{H} \mid \mathcal{E}_1, \mathcal{E}_2)$ on the values $\Pr(\mathcal{H} \mid \mathcal{E}_1)$ and $\Pr(\mathcal{H} \mid \mathcal{E}_2)$ can be anything. However if we make the additional assumption of conditional independence of the events $\mathcal{E}_1$ and $\mathcal{E}_2$ i.e.

$$\Pr(\mathcal{E}_1 \mid \mathcal{E}_2, \mathcal{H}) = \Pr(\mathcal{E}_1 \mid \mathcal{H}),$$

or equivalently $\Pr(\mathcal{E}_1, \mathcal{E}_2 \mid \mathcal{H}) = \Pr(\mathcal{E}_1 \mid \mathcal{H}) \cdot \Pr(\mathcal{E}_2 \mid \mathcal{H})$ then we may deduce that

$$\frac{\Pr(\mathcal{H} \mid \mathcal{E}_1, \mathcal{E}_2)}{\Pr(\mathcal{H})} = \tilde{K} \cdot \frac{\Pr(\mathcal{H} \mid \mathcal{E}_1)}{\Pr(\mathcal{H})} \cdot \frac{\Pr(\mathcal{H} \mid \mathcal{E}_2)}{\Pr(\mathcal{H})}$$

where $\tilde{K}$ is a constant independent of $\mathcal{H}$ and equal to

$$\tilde{K} = \frac{\Pr(\mathcal{E}_1) \cdot \Pr(\mathcal{E}_2)}{\Pr(\mathcal{E}_1, \mathcal{E}_2)}.$$

The formulas follow directly from Bayes' theorem for conditional probabilities. If we additionally assume unconditional independence of the events $\mathcal{E}_1$ and $\mathcal{E}_2$ i.e. $\Pr(\mathcal{E}_1 \mid \mathcal{E}_2) = \Pr(\mathcal{E}_1)$ then we obtain that $\tilde{K} = 1$.

Unconditional independence is neither implied by nor implies conditional independence. Accordingly unconditional independence involves an additional assumption which is often unjustified and in any case is typically unnecessary. The reason is that we usually are interested in the comparative probabilities of a sequence of propositions $\mathcal{H}_1$ $\mathcal{H}_2, \ldots$ based on the evidence $\mathcal{E}_1$ $\mathcal{E}_2$. We are interested in the comparative probabilities $\Pr(\mathcal{H}_k \mid \mathcal{E}_1, \mathcal{E}_2)$ and since the constant of proportionality $\tilde{K}$ is independent of $\mathcal{H}_k$ we are not concerned with its value. Note however that we now require a set of conditional independence assumptions one for each hypothesis of the form

$$\Pr(\mathcal{E}_1 \mid \mathcal{E}_2, \mathcal{H}_k) = \Pr(\mathcal{E}_1 \mid \mathcal{H}_k).$$

The theory is usually applied when one is given a sequence of evidence. Thus if $\mathcal{E}_1$ $\mathcal{E}_2$ $\mathcal{E}_3$ $\ldots$ is a sequence of pieces of evidence we have

$$\frac{\Pr(\mathcal{H}_k \mid \mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3 \ldots)}{\Pr(\mathcal{H}_k)} = K \cdot \prod_i \frac{\Pr(\mathcal{H}_k \mid \mathcal{E}_i)}{\Pr(\mathcal{H}_k)}$$

under the assumption that the conditional independence relations

$$\Pr(\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \ldots \mid \mathcal{H}_k) = \prod_i \Pr(\mathcal{E}_i \mid \mathcal{H}_k), \quad k = 1, 2, \ldots.$$

hold. The constant of proportionality $K$ is independent of $\mathcal{H}_k$ and equals

$$K = \frac{\prod_i \Pr(\mathcal{E}_i)}{\Pr(\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \ldots)}.$$

There are many different ways that the conditional independence relations can be satisfied. However all pairwise conditional independence relations are insufficient. Instead we need a sequence of relations. The following relations suffice:

$$\Pr(\mathcal{E}_j \mid \mathcal{E}_{j_1}, \mathcal{E}_{j_2}, \ldots, \mathcal{E}_{j_r}, \mathcal{H}_k) = \Pr(\mathcal{E}_j \mid \mathcal{H}_k), \quad r \geq 1, \quad j \notin \{j_1, j_2, \ldots, j_r\} \ . \quad (7.2)$$

That is each piece of evidence must be independent of the union of any other pieces of evidence.

It is common to use logarithms rather than the probabilities in order to turn products into sums. Thus given the conditional independence assumptions from Eqn. 7.2 we have

$$\log\left(\frac{\Pr(\mathcal{H}_k \mid \mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \ldots)}{\Pr(\mathcal{H}_k)}\right) = \log K + \sum_i \log\left(\frac{\Pr(\mathcal{H}_k \mid \mathcal{E}_i)}{\Pr(\mathcal{H}_k)}\right), \quad (7.3)$$

where the bias term $\log K$ is independent of the hypothesis $\mathcal{H}_k$. Using Bayes' theorem one more time Eqn. 7.3 may be rewritten as

$$\log\left(\frac{\Pr(\mathcal{H}_k \mid \mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \ldots)}{\Pr(\mathcal{H}_k)}\right) = \log K + \sum_i \log\left(\frac{\Pr(\mathcal{E}_i \mid \mathcal{H}_k)}{\Pr(\mathcal{E}_i)}\right) \quad (7.4)$$

which is the form in which we will use incremental probability assessment.

## 7.3 Reasoning with Parts

Suppose we are looking for a chair in the image of a room. We know that a chair consists of several parts: legs a seat and a chair-back. An appealing strategy is to search separately for legs seats and backs and then to put the parts together to find instances of chairs. This approach combines the bottom-up processing of feature extraction and simple object detection with the top-down reasoning about the components of chairs. We readily arrive at a situation where we have

many candidate legs, seats, and chair-backs. Further, it is customary to have probabilities or degrees of certainty attached to each such detection. When we combine the parts to locate instances of the chair object, it is desirable to combine the probabilities or certainty levels into a single estimate of the likelihood of each detection of a chair.

The difficulty arises from the fact that the parts of the chair are not independent. Indeed, having four candidate legs in the appropriate positions makes the likelihood of a chair much greater than the probability based on any single leg alone. We might be very unsure about any single leg, but the confluence of the four legs makes the presence of a chair at that location very likely. Conditional independence is also violated: under the assumption that the object is a chair, the presence of a leg in one spot makes the presence of a leg in other predictable locations much more probable.

If we discard positional information, then a certain level of conditional independence is restored. Under the assumption that a chair is present, the information that a leg is present tells us nothing new about the probability of the presence of another leg (or chair-back or seat), since the conditional hypothesis already tells us that these legs are present. There are two problems with this "solution." First, it is precisely this geometric information that we wish to use in order to accurately deduce the location and presence of objects. Second, we need conditional independence for all possible objects in our database. Thus, although the presence of chair legs might be conditionally independent under the assumption of the presence of a chair, under the assumption of the presence of a sofa, the presence of a chair leg makes the presence of another chair leg much more probable.

Despite these difficulties there have been successful object recognition systems that reason about parts using heuristic formulas to combine evidence that accumulates as subparts are found in appropriate positions [28 67].

We next show that in the geometric hashing situation where we have a fixed basis set and simple features the individual features in the image are conditionally independent.

## 7.4 Conditional Independence

Let us return to the formulation of object recognition as given in section 2.2.1. We will assume that a basis tuple comprising $c$ image features is chosen and fixed:

$$\mathcal{B} = \{\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_2}, \ldots, \mathbf{p}_{\mu_c}\}$$

The hypothesis is formed by a matching involving $\mathcal{B}$. That is a hypothesis will be formed by the predicate that a matching can be extended to a valid interpretation i.e. that

$$[m, [\mathbf{f}_{m,j_1}, \mathbf{p}_{\mu_1}], [\mathbf{f}_{m,j_2}, \mathbf{p}_{\mu_2}], \ldots, [\mathbf{f}_{m,j_c}, \mathbf{p}_{\mu_c}]]$$

can be extended to an interpretation involving $r$ pairs $r \gg c$. Note that a hypothesis involves the choice of a model index $m$ a selection of $c$ features from the model $(\mathbf{f}_{j_1}, \mathbf{f}_{j_2}, \ldots, \mathbf{f}_{j_c})$ and an image basis $\mathcal{B}$. Accordingly we will denote this hypothesis by

$$\mathcal{H}(m, [j_1, j_2, \ldots j_c], \mathcal{B}).$$

There is a total of $\mathcal{O}(Mn^c)$ possible hypotheses for a given fixed basis tuple $\mathcal{B}$.

The evidence for any particular hypothesis will be based on features of the image except for the basis set:

$$\mathcal{S}' = \mathcal{S} - \mathcal{B} = \{\mathbf{p}_l \mid l \notin \{\mu_1, \mu_2, \ldots, \mu_c\}\}.$$

We will only use the evidence that comes from the hash locations

$$\boldsymbol{\xi}_{\nu_1 \ldots \nu_d} = \mathbf{h}\left(\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_2}, \ldots, \mathbf{p}_{\mu_c}, \mathbf{p}_{\nu_1}, \mathbf{p}_{\nu_2}, \ldots, \mathbf{p}_{\nu_d}\right).$$

where $\{\mathbf{p}_{\nu_1}, \mathbf{p}_{\nu_2}, \ldots, \mathbf{p}_{\nu_d}\} \subset \mathcal{S}'$. The first $c$ features are all distinct whereas the final $d$ features are permitted to have repetitions. There are potentially other pieces of evidence available for example from permutations of the arguments to the hash function but we will not use such permutations and depending on the symmetries exhibited by the hash functions such information will usually be redundant.[3] Thus the information is formed from the hash function applied to the fixed basis tuple together with $d$-tuples of features in the image.[4] When $d = 1$ this means that there are $S - c$ pieces of evidence. In general there are $(S-c)^d$ pieces of evidence available assuming that all permutations of the arguments $\mathbf{p}_{\nu_1}, \mathbf{p}_{\nu_2}, \ldots, \mathbf{p}_{\nu_d}$ yield extra information. Each piece of evidence can be regarded as a realization of a random variable that can be defined as

$$\mathbf{Y} = \mathbf{h}\left(\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_2}, \ldots, \mathbf{p}_{\mu_c}, \mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_d\right)$$

where the various $\mathbf{X}_i$ are independent random vectors assuming values in $\mathcal{S}'$.[5] When we speak of conditional independence of the evidence we refer to information that comes from realizations of this random vector and not to the independence of this random vector from other random processes.

In order to deduce that the evidence is conditionally independent we will need one further assumption. We need to assume that the entries in the space of invariants do not "clump " so that no two entries of the same model and with a

[3] For example if $\mathbf{h}$ is transformation-specific then permutations carry no added information.
[4] Recall that $d$ is the number of extra features.
[5] Note that it can happen that some of the $\mathbf{X}_i$ realize the same feature.

fixed basis set (i.e. the same first $c$ model features) land near one another in the space of invariants. Ideally the hash function is injective on the $d$-tuple of features for a fixed basis tuple in the first $c$ parameters so that the condition amounts to saying that model features are distinct and separated. This is typically the case for a transformation-specific hash function. If more than one model features land in approximately the same location then the features should be coalesced into a single feature or the hash function should be modified.

We claim that the evidence is conditionally independent. In order to see this let $\mathcal{H}(i,[j_1,j_2,\ldots,j_c],\mathcal{B})$ be a hypothesis. The evidence is the set of hash locations

$$\Gamma = \left\{ \boldsymbol{\xi}_{\nu_1,\nu_2,\ldots,\nu_d} \mid \{\mathbf{p}_{\nu_1},\mathbf{p}_{\nu_2},\ldots,\mathbf{p}_{\nu_d} \subset \mathcal{S}'\} \right\} .$$

To show independence we consider the probability of a hash to a given location under the assumption of the hypothesis $\mathcal{H}(i,[j_1,j_2,\ldots j_k],\mathcal{B})$. We are also interested in the probability of a hash to the same location under the same hypothesis given a subcollection $\Gamma' \subset \Gamma$ of other hash locations. Since the hash value locations are continuous variables the probabilities can be replaced by evaluations of the corresponding probability density functions and we will denote the two functions by

$$f_{\mathbf{Y}|\mathcal{H}}(\boldsymbol{\xi}) \quad \text{and} \quad f_{\mathbf{Y}|\Gamma',\mathcal{H}}(\boldsymbol{\xi}),$$

respectively; $\mathcal{H}$ is an abbreviation for $\mathcal{H}(i,[j_1,j_2,\ldots j_q],\mathcal{B})$.

Conditional independence of the evidence means the following: given a single hash point $\boldsymbol{\xi}_{\nu_1,\nu_2,\ldots,\nu_d}$ together with a collection $\Gamma'$ of other hash locations all distinct then

$$f_{\mathbf{Y}|\mathcal{H}}(\nu_1,\nu_2,\ldots\nu_d) = f_{\mathbf{Y}|\Gamma',\mathcal{H}}(\nu_1,\nu_2,\ldots\nu_d).$$

Let us first consider $f_{\mathbf{Y}|\mathcal{H}}\left(\boldsymbol{\xi}_{\nu_1,\nu_2,\ldots,\nu_d}\right)$. From our assumption we know that the $m$-th model is embedded in the image with model features $\mathbf{f}_{j_1}$ $\mathbf{f}_{j_2}$ $\ldots$ $\mathbf{f}_{j_c}$ mapping to the image points $\mathbf{p}_{\mu_1}$ $\mathbf{p}_{\mu_2}$ $\ldots$ and $\mathbf{p}_{\mu_c}$. Although some of the remaining points of model $m$ may be obscured there is a strong likelihood of their presence and thus there is a large likelihood of obtaining hash values at locations corresponding to hashes of model $m$ using basis $\mathbf{f}_{m,j_1}$ $\mathbf{f}_{m,j_2}$ $\ldots$ $\mathbf{f}_{m,j_c}$. Indeed at every location in the hash table where there is an entry with tag $[m,[j_1,j_2,\ldots,j_q]]$ there is a large likelihood of an image hash to the same location (since the hash function is transformation invariant and the corresponding model features occur in the image). Thus the probability density function for $\mathbf{h}\left(\mathbf{p}_{\mu_1},\mathbf{p}_{\mu_2},\ldots,\mathbf{p}_{\mu_c},\mathbf{X}_1,\mathbf{X}_2,\ldots,\mathbf{X}_d\right)$ will include spikes at the positions where these entries occur in the hash table. The spikes will be smeared out due to the possibility of noise in the image features. All the other hashes of $\mathbf{h}\left(\mathbf{p}_{\mu_1},\mathbf{p}_{\mu_2},\ldots,\mathbf{p}_{\mu_c},\mathbf{X}_1,\mathbf{X}_2,\ldots,\mathbf{X}_d\right)$ will involve one or more of the random vectors $\mathbf{X}_i$ taking on the value of image features not belonging to the model $m$. The contribution to the density function due to these hashes will be more continuous and depends upon the background distribution of features in the image. Typically we will assume that this background distribution is uniform in the image although in applications one might be able to solve analytically or empirically for the true distribution. In any case the distribution will add to the spikes in the distribution due to the model features embedded in the image. The resulting distribution will be a weighted superposition of the two distributions appearing in Figures 7.3 and 7.4; as we will show in what follows an analytic expression can derived for this distribution.

Next consider the value of the density function $f_{\mathbf{Y}|\Gamma',\mathcal{H}}\left(\boldsymbol{\xi}_{\nu_1,\nu_2,\ldots,\nu_d}\right)$. We are

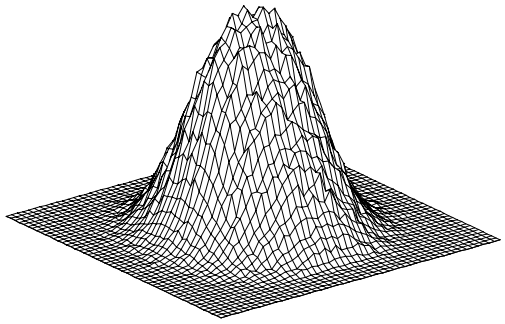

**Figure 7.3 The probability density function of hashes in the space of invariants which are generated by image features not belonging to the model that is embedded in the image.**

given a collection of points $\Gamma'$ in the space of invariants and we know that the image has generated hashes to these locations. We also know that the model $m$ is embedded in the image with a known transformation. We are given a new hash location $\boldsymbol{\xi}_{\nu_1,\nu_2,\ldots,\nu_d}$ which is generated by a distinct hash and we ask whether knowledge of $\Gamma'$ changes the likelihood of the hash to this location.

Some of the hashes in $\Gamma'$ may occur at locations that confirm the presence of the model $m$ occurring at or near locations where the spikes occur in the density function $f_{\mathbf{Y}|\mathcal{H}}$ due to hashes involving only point features from the embedded model. However since we already know from the hypothesis that the model is present this information tells us nothing. Each such hash makes it much less likely that another hash will occur at the same location since the combination of image features from the model generating the hash is thereby "used." However since the hash $\boldsymbol{\xi}_{\nu_1,\nu_2,\ldots,\nu_d}$ is known not to be among the hashes in $\Gamma'$ and since

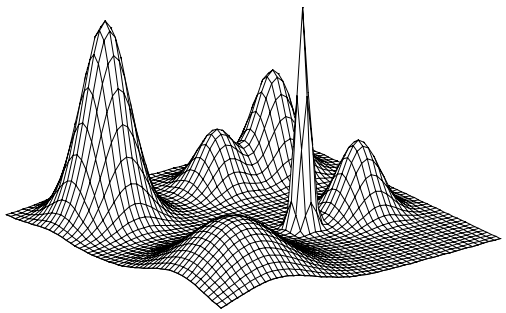


**Figure 7.4 The probability density function of the hashes generated by the features of a model that is embedded in an image. In this example, n–c=6.**

hashes due to model points are separated the suppression of the likelihood at hash location entries will not influence the density function evaluated at $\xi_{\nu_1,\nu_2,\ldots,\nu_d}$. The remaining information from $\Gamma'$ includes hashes from features that are not part of model $m$ which tell us nothing.[6] Accordingly the likelihood of the hash to location $\xi_{\nu_1,\nu_2,\ldots,\nu_d}$ is unchanged by knowledge of the hashes to locations in $\Gamma'$ which is to say that the evidence in the hash locations $\Gamma$ are independent under the hypothesis $\mathcal{H}$ for all possible hypotheses.

[6] We could potentially use the hashes from $\Gamma'$ to deduce the locations of some image features providing there are enough hashes that utilize the same image features. Once we know the value of other image features then the density distribution is modified either adding spikes at positions or blurred submanifolds where hashes are somewhat more likely. However extracting this information would require a sophisticated statistical analysis of $\Gamma'$ and also likely depends on a high dimensionality of the space of invariants. This dependence does not occur when $d = 1$ which is our application. We would be inclined to discount this dependence in other cases also but a more detailed analysis is required.

## 7.5 Density Functions

In order to develop the Bayesian geometric hashing algorithm we will need to make use of the density functions

$$f_{\mathbf{Y}|\mathcal{H}}(\boldsymbol{\xi}) \quad \text{and} \quad f_{\mathbf{Y}}(\boldsymbol{\xi}).$$

Recall that $\mathbf{Y}$ is the random vector $\mathbf{h}(\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_2}, \ldots, \mathbf{p}_{\mu_c}, \mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_d)$ where the $\mathbf{X}_i$ independently take on image feature values in $\mathcal{S}'$. That is we must compute the density of hashes in the space of invariants obtained by applying the hash function to the concatenation of the fixed basis tuple with arbitrary $d$-tuples of image features (chosen from $\mathcal{S}'$) both with and without the assumption that the chosen basis participates in a valid interpretation of a model object. The first probability density function was discussed in the previous subsection whereas the second function is new.

We next provide more precise general formulas. In applications the density functions will have to be computed using specific noise models. Our treatment will assume that the map

$$\mathcal{T}(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_d) = \mathbf{h}(\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_2}, \ldots, \mathbf{p}_{\mu_c}, \mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_d) \tag{7.5}$$

is a one-to-one differentiable map from $d$-tuples of image features to the space of invariants and that the Jacobian of the transformation is nowhere singular. In this case $\mathbf{h}$ is necessarily $F$-specific but we are assuming more. Extensions to more general cases are possible.

Consider a point

$$\boldsymbol{\xi} \equiv \mathcal{T}(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_d) = \mathbf{h}(\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_2}, \ldots, \mathbf{p}_{\mu_c}, \mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_d).$$

Using the stochastic independence of the $\mathbf{X}_i$'s, the probability density function of $\mathbf{Y}$ evaluated at $\boldsymbol{\xi}$ can be written as

$$f_{\mathbf{Y}}(\boldsymbol{\xi}) = \frac{\prod_{i=1}^{d} f_{\mathbf{X}}(\mathbf{r}_i)}{\mid J_{\mathcal{T}}(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_d) \mid} .$$

Here $J_{\mathcal{T}}$ is the Jacobian matrix of the transformation $\mathcal{T}$ expressed as a function of $\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_d$ (see Eqn. 7.5), and $f_{\mathbf{X}}$ is the probability density function for the set $\mathcal{S}'$ of image features. A similar expression holds for $f_{\mathbf{Y}|\mathcal{H}}(\boldsymbol{\xi})$:

$$f_{\mathbf{Y}|\mathcal{H}}(\boldsymbol{\xi}) = \frac{\prod_{i=1}^{d} f_{\mathbf{X}|\mathcal{H}}(\mathbf{r}_i)}{\mid J_{\mathcal{T}}(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_d) \mid} .$$

In this expression $f_{\mathbf{X}|\mathcal{H}}$ is the probability density function of the set $\mathcal{S}'$ of image features, under the hypothesis $\mathcal{H}(m, [j_1, j_2, \ldots, j_c], \mathcal{B})$, which will imply the likely existence of image points at certain locations corresponding to points of the model $m$.

In the case where $\mathcal{T}$ is not a one-to-one mapping of $d$-tuples of features into the space of invariants, then the revised formulas involve integrals on the right hand sides over submanifolds of a $d$-fold feature space, where $d$-tuples of features map to the given hash location.

Typically, we will assume that feature values are continuous and that all parameter values are equally likely. For example, when image features are points in a rectangular image, we will assume a uniform density distribution over the image domain. Thus $f_{\mathbf{X}}(\mathbf{r})$ is often a constant over the range of features values.

For $f_{\mathbf{X}|\mathcal{H}}$, we typically have a mix of the background distribution $f_{\mathbf{X}}$ and a set of *smeared* delta masses at the predicted positions for model points:

$$f_{\mathbf{X}|\mathcal{H}}(\mathbf{r}) = (1-\beta) f_{\mathbf{X}}(\mathbf{r}) + \frac{\beta}{n-c} \cdot \sum_{j=1}^{n-c} g_{\sigma}(\mathbf{r} - \mathbf{q}_j) ,$$

where $g_\sigma(\cdot)$ is a Gaussian distribution representing the likely distribution of feature values around the image features $\mathbf{q}_j$ that are predicted by hypothesis $\mathcal{H}$ from model $m$. Note that there are $n-c$ predicted feature vectors for features in $\mathcal{S}'$ from model $m$ representing the features of the model less the basis set $\{\mathbf{f}_{m,j_1}, \mathbf{f}_{m,j_2}, \ldots, \mathbf{f}_{m,j_c}\}$ as embedded in the image. The coefficient $\beta$ represents a weighting factor that models the number of features from the model $m$ which are expected to actually occur in the image features (i.e. $\mathcal{S}'$) as compared to the total number of image features in $\mathcal{S}'$. For a given image feature $\mathbf{r}$ at most one model feature in model $m$ can be "close" to $r$. If we denote the closest such feature by $\mathbf{q}_{j(\mathbf{r})}$ then since $g_\sigma$ typically falls off rapidly the expression for $f_{\mathbf{X}|\mathcal{H}}(\mathbf{r})$ can be simplified:

$$f_{\mathbf{X}|\mathcal{H}}(\mathbf{r}) \approx (1-\beta) f_{\mathbf{X}}(\mathbf{r}) + \frac{\beta}{n-c} \cdot g_\sigma\left(\mathbf{r} - \mathbf{q}_{j(\mathbf{r})}\right).$$

In this case the resulting formula for $f_{\mathbf{Y}|\mathcal{H}}$ will have the form:

$$f_{\mathbf{Y}|\mathcal{H}}(\boldsymbol{\xi}) \approx \frac{\prod_{i=1}^{d}\left[(1-\beta) f_{\mathbf{X}}(\mathbf{r}_i) + \frac{\beta}{n-c} \cdot g_\sigma\left(\mathbf{r}_i - \mathbf{q}_{j(\mathbf{r}_i)}\right)\right]}{\mid J_{\mathcal{T}}(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_d) \mid}.$$

Qualitatively we see that the product is small unless all or most of the $\mathbf{r}_i$ are near model features in the image. If they are all near model features then the location of $\boldsymbol{\xi}$ is near an entry $\omega$ with $\omega = [m, [j_1, j_2, \ldots, j_c, j(\mathbf{r}_1), j(\mathbf{r}_2), \ldots, j(\mathbf{r}_d)]]$. Thus we expect peaks in $f_{\mathbf{Y}|\mathcal{H}}(\boldsymbol{\xi})$ near entries having tags $[m, [j_1, j_2, \ldots, j_c]]$. There are other "surfaces" in the space of invariants where $f_{\mathbf{Y}|\mathcal{H}}$ will be similarly elevated due to all but one or all but a few of the $\mathbf{r}_i$'s falling close to model features $\mathbf{q}_j$'s. These surfaces only appear when $d > 1$ and can most likely be ignored for most applications. Whether the surfaces can be useful for object recognition as described in the next subsection is to our knowledge unexplored. Of course

this formula is only an example and different applications will result in different models for $f_{\mathbf{X}|\mathcal{H}}$ and thus different formulas for $f_{\mathbf{Y}|\mathcal{H}}$.

We may summarize our exploration of the conditional density function $f_{\mathbf{Y}|\mathcal{H}}$ by noting that we have given evidence for a decomposition theorem which says that $f_{\mathbf{Y}|\mathcal{H}}$ may be written as:

$$\begin{aligned} f_{\mathbf{Y}|\mathcal{H}}(\boldsymbol{\xi}) = \quad & \beta_1 f_{\mathbf{Y}}(\boldsymbol{\xi}) + \beta_2 \mathbb{S}(\boldsymbol{\xi}) \\ & + \beta_3 \sum_{k_1 \in \mathcal{I}} \cdots \sum_{k_d \in \mathcal{I}} \tilde{g}\left(\boldsymbol{\xi} - \boldsymbol{\zeta}\left([m, [j_1, j_2, \ldots, j_c, k_1, k_2, \ldots, k_d]]\right)\right) . \end{aligned}$$

Here $\mathbb{S}$ is a function which is large only in the "surfaces" described above and is generally neglected and $\mathcal{I}$ is the set of indices of $m$'s model features that have not been used in the model basis chosen by $\mathcal{H}$. I.e. $\mathcal{I} = \{1, 2, \ldots, n\} - \{j_1, j_2, \ldots, j_c\}$. In the third term $\tilde{g}$ is a local density function that accounts for a smeared spike at the location of the entries $\omega = [m, [j_1, j_2, \ldots, j_c, k_1, k_2, \ldots, k_d]]$ which is the set of all entries in the hash table generated by model $m$ and the basis $[j_1, j_2, \ldots, j_c]$. We recall that the location of the entry is denoted by $\boldsymbol{\zeta}(\omega)$. The density function $\tilde{g}$ will depend on the hypothesis $\mathcal{H}$ and the error model for errors in the image features. For the case where $\mathcal{T}$ is one-to-one we have that

$$\tilde{g}\left(\mathcal{T}(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_d)\right) = \frac{\prod_{i=1}^{d} g_\sigma\left(\mathbf{r}_i - \mathbf{q}_{j(\mathbf{r}_i)}\right)}{\left| J_{\mathcal{T}}(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_d) \right|} .$$

We further note in passing that when $\mathcal{T}$ is one-to-one and smooth as we have assumed here it is possible to translate the density function measurements to feature space (as opposed to space of invariants) and thus measure the log-density ratio

$$\log\left(\frac{f_{\mathbf{Y}|\mathcal{H}}(\boldsymbol{\xi})}{f_{\mathbf{Y}}(\boldsymbol{\xi})}\right)$$

in terms of a sum of deviations (measured relative to $g_\sigma$) in feature space over the $d$-tuple of features $\mathbf{p}_{\nu_1}\ \mathbf{p}_{\nu_2}\ \ldots \mathbf{p}_d$. It is this log-density ratio that we will use for the Bayesian geometric hashing algorithm.

Note that the probability density function $f_{\mathbf{Y}}$ can potentially depend on the image basis $\mathcal{B} = \{\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_2}, \ldots, \mathbf{p}_{\mu_c}\}$ due to the $\mid J_{\mathcal{T}} \mid$ term in the denominator of the defining equation. Alternatively it is possible to use the same density function $f_{\mathbf{Y}}$ independent of the basis $\mathcal{B}$: more specifically we can compute either empirically or analytically the average density function over all possible basis selections (see section 7.10). In some cases use of a basis-dependent probability density function actually simplifies the formulas (see section 7.9).

## 7.6 Bayesian Geometric Hashing

We may now piece the various components together. Using Eqn. 7.4 we have that

$$\log\left(\Pr\left(\mathcal{H} \mid \Gamma\right)\right) = \log K + \log\left(\Pr\left(\mathcal{H}\right)\right) + \sum_{\nu_1} \ldots \sum_{\nu_d} \log\left(\frac{f_{\mathbf{Y}|\mathcal{H}}\left(\boldsymbol{\xi}_{\nu_1,\nu_2,\ldots,\nu_d}\right)}{f_{\mathbf{Y}}\left(\boldsymbol{\xi}_{\nu_1,\nu_2,\ldots,\nu_d}\right)}\right).$$

Recall that $\Gamma$ is the set of hashes that arise from the image features $\mathcal{S}'$ and that $\boldsymbol{\xi}_{\nu_1,\nu_2,\ldots,\nu_d}$ are the hash locations of those hashes. Since we are typically only concerned with the relative strength of the probability of a given hypothesis in order to find the top few hypotheses with the maximum support and since we will typically assume that all hypotheses are equally likely we may neglect the first two terms which are independent of $\mathcal{H}$ and simply compute the sum in order to obtain total support weights $Z\left(\mathcal{H}\right)$ for each hypothesis. We thus have

$$Z\left(\mathcal{H}\right) = \sum_{\nu_1} \ldots \sum_{\nu_d} \log\left(\frac{f_{\mathbf{Y}|\mathcal{H}}\left(\boldsymbol{\xi}_{\nu_1,\nu_2,\ldots,\nu_d}\right)}{f_{\mathbf{Y}}\left(\boldsymbol{\xi}_{\nu_1,\nu_2,\ldots,\nu_d}\right)}\right). \tag{7.6}$$

As written the formula is not of much help since the computational cost will be immense. The formula says that for every probe using a fixed basis $\mathcal{B}$ of image features we compute hash locations using $d$-tuples of image features and with each hash location increment accumulators for every hypothesis. Recall that $\mathcal{H} = \mathcal{H}(m, [j_1, j_2, \ldots, j_c], \mathcal{B})$. Since there are $\mathcal{O}(Mn^c)$ hypotheses and we may have to perform $\mathcal{O}(S^c)$ probes we achieve no speedup over classical search methods.

However if we make use of our "decomposition theorem" for $f_{\mathbf{Y}|\mathcal{H}}$ ignoring the surface masses we then obtain the following expression for $Z(\mathcal{H})$:

$$\sum_{\nu_1} \cdots \sum_{\nu_d} \log\left(\beta_1 + \beta_3 \frac{\sum_{k_1 \in \mathcal{I}} \cdots \sum_{k_d \in \mathcal{I}} \tilde{g}\left(\boldsymbol{\xi}_{\nu_1,\nu_2,\ldots,\nu_d} - \boldsymbol{\zeta}([m, [j_1, j_2, \ldots, j_c, k_1, \ldots, k_d]])\right)}{f_{\mathbf{Y}}\left(\boldsymbol{\xi}_{\nu_1,\nu_2,\ldots,\nu_d}\right)}\right).$$

Recall that $\mathcal{I}$ is the set of available indices for feature points in the computation of an entry corresponding to model $m$ and basis $[j_1, j_2, \ldots, j_c]$.

Consider a fixed $\mathcal{H}$. For each hash $\boldsymbol{\xi}_{\nu_1,\nu_2,\ldots,\nu_d}$ there is a contribution to $Z(\mathcal{H})$. If the hash does not fall near any of the entries with tag equal to $[m, [j_1, j_2, \ldots, j_c]]$ (i.e. a tag associated with the hypothesis $\mathcal{H}$) then all $\tilde{g}$ terms may be ignored and the contribution is $\log(\beta_1)$. If the hash falls near one such entry then by assumption it will fall near *only one* such entry. This is because for a given model/basis $[m, [j_1, j_2, \ldots, j_c]]$ the corresponding entries are separated and so that a single hash can be located in the vicinity of at most one such entry (although there may be multiple entries in the neighborhood each with a different tag and thus contributing to different hypotheses). Suppose that for the tag $[m, [j_1, j_2, \ldots, j_c]]$ the entry $\omega$ is in the neighborhood of the hash $\boldsymbol{\xi}_{\nu_1,\nu_2,\ldots,\nu_d}$. Then

the contribution to $\mathcal{H}$ will be

$$\log\left(\beta_1 + \beta_3 \frac{\tilde{g}\left(\boldsymbol{\xi}_{\nu_1,\nu_2,\ldots,\nu_d} - \boldsymbol{\zeta}(\omega)\right)}{f_{\mathbf{Y}}\left(\boldsymbol{\xi}_{\nu_1,\nu_2,\ldots,\nu_d}\right)}\right).$$

We see that the criterion for a hash to be near an entry is that the value of the second term of the log is significant relative to the first term. The criterion may be reinterpreted in terms of a threshold for $\tilde{g}$ for particular applications.

The weighted voting may thus be organized as follows. Initially every entry has a weight $z(\omega)$ equal to zero. For each hash $\boldsymbol{\xi}_{\nu_1,\nu_2,\ldots,\nu_d}$ we use a $k-D$ ($k$-Dimensional) tree or similar data structure to access entries in the neighborhood. The weight

$$\begin{aligned} z(\omega) &= \log\left(\beta_1 + \beta_3 \frac{\tilde{g}\left(\boldsymbol{\xi}_{\nu_1,\nu_2,\ldots,\nu_d} - \boldsymbol{\zeta}(\omega)\right)}{f_{\mathbf{Y}}\left(\boldsymbol{\xi}_{\nu_1,\nu_2,\ldots,\nu_d}\right)}\right) - \log(\beta_1) \\ &= \log\left(1 + \frac{\beta_3}{\beta_1} \frac{\tilde{g}\left(\boldsymbol{\xi}_{\nu_1,\nu_2,\ldots,\nu_d}\right) - \boldsymbol{\zeta}(\omega)}{f_{\mathbf{Y}}\left(\boldsymbol{\xi}_{\nu_1,\nu_2,\ldots,\nu_d}\right)}\right) \end{aligned}$$

is computed and applied to $\omega$. If the entry already has a nonzero weight assigned to it from a previous hash then we have two distinct collections of $n$ scene features vying for correspondence to a single group of $n$ features in a model. Since only one of the two matchings can be correct and we should not allow both to independently support the matching hypothesis $\mathcal{H}$ we assign to $z(\omega)$ the maximum of the values computed from the competing hashes.

After all hashes are processed we compute total support values $Z(\mathcal{H})$ by accumulating weights for entries with common tags. However to account for the hashes that do not fall near entries with the given tag each bucket begins with a bias term equal to the number of hashes times $\log(\beta_1)$. Assuming there are

$(S-c)^d$ hashes we have

$$Z(\mathcal{H}) = (S-c)^d \log(\beta_1) + \sum_{k_1 \in \mathcal{I}} \cdots \sum_{k_d \in \mathcal{I}} z([m, [j_1, j_2, \ldots, j_c, k_1, k_2, \ldots, k_d]]).$$

We see that the net effect is that each hash implicitly contributes $\log(\beta_1)$ through the bias term or it contributes the same implicit amount together with the explicit $z(\omega)$ through a nearby entry. Since the $z(\omega)$ contains a term that cancels the $\log(\beta_1)$ amount the hash contributes the correct quantity

$$\log\left(\beta_1 + \beta_3\left(\frac{\tilde{g}(\boldsymbol{\xi}-\boldsymbol{\zeta})}{f_{\mathbf{Y}}(\boldsymbol{\xi})}\right)\right).$$

In practice since a constant amount added to each hypothesis is irrelevant (we only seek the top few hypothesis) we may omit the bias quantity in the computation and simply set

$$Z(\mathcal{H}) = \sum_{k_1 \in \mathcal{I}} \cdots \sum_{k_d \in \mathcal{I}} z([m, [j_1, j_2, \ldots, j_c, k_1, k_2, \ldots, k_d]]).$$

Quite simply all entries with the same tag combine their $z(\omega)$ values to yield a degree of support for the corresponding hypothesis.

## 7.7 Exact versus Approximate Matching

The hypothesis $\mathcal{H}(m, [j_1, j_2, \ldots, j_c], \mathcal{B})$ represents the statement that model $m$ is present in the image with the matching defined by placing the basis set of model features $\mathbf{f}_{m,j_1}$ $\mathbf{f}_{m,j_2}$ $\ldots$ $\mathbf{f}_{m,j_c}$ respectively in correspondence with the features in $\mathcal{B}$. In the way we have formulated the hypothesis the transformation is uniquely and precisely defined by the pairing of the features. There are two reasons why the hypothesis might be rejected.

- First, it may not be true that the matching is the correct one, and that $\mathcal{B}$ is not part of an instance of model $m$ with the appropriate association of model features as given in the hypothesis;

- Second, there might be too much noise in the image features represented by the chosen $\mathcal{B}$, so that the transformation defined by pairing model points $\mathbf{f}_{m,j_1}$, $\mathbf{f}_{m,j_2}$, ... $\mathbf{f}_{m,j_c}$ respectively with $\mathcal{B}$ in the image does not bring sufficiently closely into correspondence features in $m$ and $\mathcal{S}'$. This can happen even though there does exist a transformation which *approximately* brings into correspondence model points $\mathbf{f}_{m,j_1}$, $\mathbf{f}_{m,j_2}$, ... $\mathbf{f}_{m,j_c}$ and $\mathcal{B}$, together with other features of model $m$ with features in $\mathcal{S}'$, establishing a valid interpretation of model $m$. In this case, the transformation will carry the model basis to features that lie in the neighborhood of the respective features of $\mathcal{B}$.

If we use the probe-based search strategy of locating models in the image, then it can happen that we chose a collection of features $\mathcal{B}$ as a basis, and that this basis belongs to a proper interpretation of a model, but that the model will not be recognized, due to error in features in $\mathcal{B}$. We identify three ways of dealing with this situation.

First, the problem may be ignored. Then, some valid hypotheses may be rejected. However, the failure of a given probe with one basis $\mathcal{B}$ may be rescued by a subsequent probe with a different basis $\mathcal{B}'$. We must analyze the probability that a given valid basis will yield a rejection of the corresponding hypothesis, and then the probability that most or all bases on a given embedded model will lead to rejections. We do not conduct such an analysis here, but we note that generally the best compromise matching will place some subset of model features

(generally at least a basis set) close to corresponding image features.

A second alternative is to modify the hypotheses so that each hypothesis represents an approximate match between the chosen basis and a model/basis combination. That is, we can modify the hypothesis to say that $\mathcal{H}$ implies an approximate matching. Figure 7.5 depicts the difference between exact-matching hypotheses and approximate-matching hypotheses. In the computation of $\mathbf{f}_{\mathbf{X}|\mathcal{H}}$ the

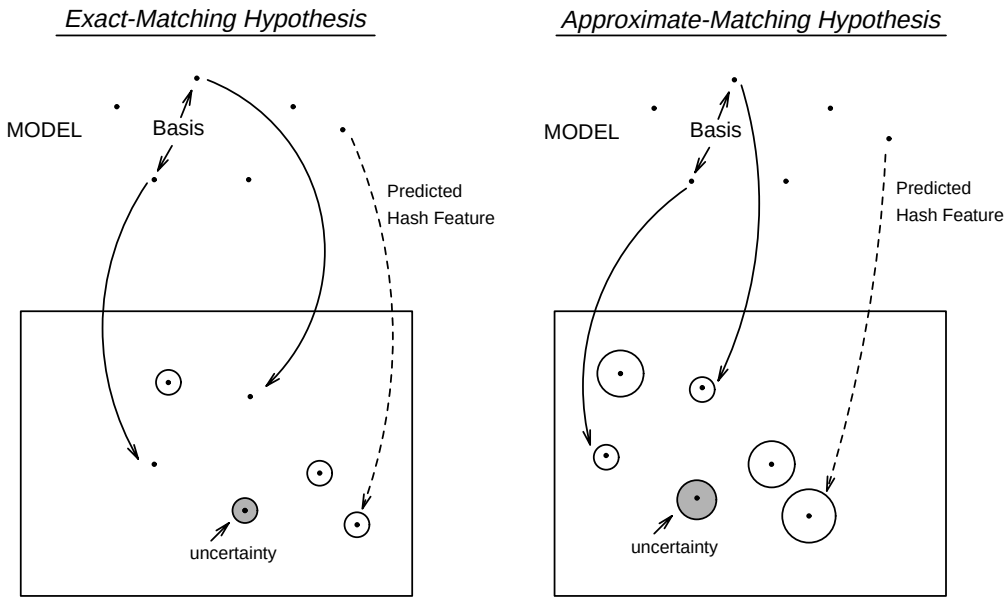


**Figure 7.5 An exact-matching hypothesis as compared to an approximate-matching hypothesis. Note that in the case of an approximate-matching hypothesis, there is a greater range of uncertainty in the predicted image features that arise as a result of the remaining features of the model.**

probability density distribution of features predicted by the presence of model $m$ under an *approximate* matching of basis $\mathbf{f}_{m,j_1}$ $\mathbf{f}_{m,j_2}$ ... $\mathbf{f}_{m,j_c}$ with image features $\mathcal{B}$ must take into account potential noise in the features predicted to be located in the image *as well as the possible noise in the basis features* $\mathcal{B}$. This

is in fact the analysis that we carried out in chapter 6. The result is that in $\mathbf{f}_{\mathbf{Y}|\mathcal{H}}$ which gives the distribution function in the space of invariants based on $\mathbf{h}(\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_2}, \ldots, \mathbf{p}_{\mu_c}, \mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_d)$ the spikes due to the predicted locations of features will be more spread out. In terms of the Eqn. 7.6 this means that the local density function $\tilde{g}$ used to construct spikes in the space of invariants will have a larger support. Approximate-matching hypotheses leave a number of free continuous parameters to be determined. When there are continuous parameters that cannot be constrained by a finite collection of possibilities this is when filtering strategies or Hough transform methods become more appropriate. We are thus led to suggest a third alternative for dealing with the possibility of rejection of valid exact-matching hypotheses.

In the third alternative the hypotheses represent approximate transformations and the density functions reflect the expanded uncertainty due to potential error in the basis features. However instead of simply voting for the probability of a hypothesis based on a nearby entry in the space of invariants we can instead vote for ranges of parameter values that are consistent with the observed positions of the feature values. The search may be organized differently than probes with basis sets. In general if a hash $\mathbf{h}(\mathbf{p}_{\nu_1}, \mathbf{p}_{\nu_2}, \ldots, \mathbf{p}_{\nu_n})$ lands near an entry at location $\mathbf{h}(\mathbf{f}_{m,j_1}, \mathbf{f}_{m,j_2}, \ldots, \mathbf{f}_{m,j_n})$ we may vote in a Hough space for a range of parameters that reasonably map the model features to corresponding image features and store the table with a hypothesis concerning the particular model. This is for example the approach of Guéziec [40] for matching $3D$ curves. We might instead have a separate Hough table for each approximate matching and use votes for local parameter values to establish probable deformations of the basis features from the observed values. Methods combining geometric hashing

with parameter voting (Hough transforms) are only beginning to be studied and extensions of the Bayesian interpretation to the case where parameter spaces and Hough transforms are involved is not considered here.

Experiments with both random perturbations of synthetic data and an actual database of real-world objects indicated that the use of approximate hypotheses leads to better results.

## 7.8 False Alarm Rates

We may now comment on false alarm rates under varying scenarios. We first suppose that the hash function is $\mathcal{F}$-specific and that exact-matching hypotheses are used. Further, to simplify things, let us assume that the number of excess features in the hash function, $d$, is one.

If neighborhood voting is used, and if a hypothesis receives $n'$ votes during a probe, then we can be certain that when the model basis of the hypothesis is put into correspondence with the basis of scene features from the probe, then exactly $n'$ features fall within a radius as defined by the neighborhood radius of the voting scheme. Ideally, the radius for the neighborhood voting scheme in the space of invariants has been adjusted, for the probe, according to the chosen basis set, so that the guarantee is that $n'$ features fall within a one- or two- (or a predetermined multiple of) sigma distance of the predicted values for the feature.

If instead weighted voting is used, then because there is a maximum weighted vote $w(\boldsymbol{\zeta}, \boldsymbol{\xi})$ that any given hash value can give to a hypothesis, we can give a guarantee of a minimum number of features from the corresponding model falling within a given radius of the predicted values, if the total vote for a hypothesis is $Z(\mathcal{H})$. Further, if the appropriate Bayesian formula is used, $Z(\mathcal{H})$ will be an

accurate measure related to the true *a posteriori* probability of the hypothesis. If the hash function is $\mathcal{F}'$-specific where $\mathcal{F}'$ is a transformation class that strictly contains $\mathcal{F}$ then all of the same comments apply but the postprocessing step will have to disambiguate between models that match with an $\mathcal{F}$ transformation and those that require the generality of an $\mathcal{F}'$ transformation.

In this sense with exact-matching hypotheses and an $\mathcal{F}$-specific hash function there can be no false alarms. The situation is more complicated with hash functions involving $d$-tuples of features appended to basis sets with $d > 1$ but similar guarantees can be given as a result of the computation of a support value for each hypothesis. These assure us that the measure of a certain value for $Z(\mathcal{H})$ will yield a valid interpretation; a simpler argument suffices to show that the existence of a valid interpretation under an exact-matching hypothesis will yield a guaranteed level of support even if the non-basis features match only within say one sigma (on the average).

As indicated above the use of the exact-matching hypotheses can lead to a hypothesis being rejected even though an approximate matching of the basis of the model to the scene basis can be extended to a valid interpretation.

If we use approximate-matching hypotheses and neighborhood voting then the guarantee that we can give for a total vote of $n'$ is much less. Each of the votes in the total count of $n'$ indicates that there exists a transformation taking the model basis together with the $(d = 1)$ extra feature of the model to within one-sigma (or an appropriate radius) of existing features in the scene. However in the absence of a Hough space to measure deviations of the basis matching there is no guarantee that the $n'$ votes correspond to the *same* transformation. We can only be guaranteed that there are $n'$ different transformations each similar

to the others in the sense that the basis set of the model is mapped to within a specified radius of the scene basis features such that remaining model features are individually approximated by scene features.

If weighted voting is used in conjunction with approximate-matching hypotheses then we once again obtain a support value which can be used to determine a minimum number of existing points such that an approximating transformation may be found for each such that the basis points of the model are mapped to the vicinity of the scene basis. However we are not guaranteed that the multiple transformations are the same. Indeed in terms of the Bayesian analysis the evidence is no longer independent due to the approximate-matching hypothesis. The difficulty is that the error in the basis features can be deduced from a sufficient number of corroborating hashes involving features from the model embedded in the scene. The net effect of the use of Bayesian-based geometric hashing with approximate matching hypotheses is the same as exact matching hypotheses but with larger error bounds on the unpaired features. Due to the lack of independence we cannot say that the values give a precise measure of the relative *a posteriori* probabilities. We will nonetheless use the computations implied by the exact-matching hypotheses for the approximate-matching case using the larger error bounds in order to be able to detect models when the basis set matches only approximately justifying the weighted-voting formulas empirically.

## 7.9 The Formulas: Exact Matching

We now apply the abstract theory. We are concerned with the matching of patterns of point features in $2D$. Thus a point feature $\mathbf{p}$ is specified by a coordinate pair $\mathbf{p} = (x, y)$. In this section we treat the case of exact-matching hypotheses

as a preparation for the next section, where the approximate-matching formulas are developed.

We will consider two classes of transformations: similarity and affine. Respectively, a basis tuple is defined by two or three points. The hash functions, which in each case operate on a basis set plus one feature, are for similarity invariance ($c = 2$):

$$\mathbf{h}_s(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3) = \begin{pmatrix} u \\ v \end{pmatrix},$$

$$\begin{pmatrix} x_2 - x_1 & -y_2 + y_1 \\ y_2 - y_1 & x_2 - x_1 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} x_3 - (x_1 + x_2)/2 \\ y_3 - (y_1 + y_2)/2 \end{pmatrix},$$

and, for affine invariance ($c = 3$):

$$\mathbf{h}_a(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4) = \begin{pmatrix} u \\ v \end{pmatrix},$$

$$\begin{pmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} x_4 - (x_1 + x_2 + x_3)/3 \\ y_4 - (y_1 + y_2 + y_3)/3 \end{pmatrix}.$$

Clearly, we have used the notation $\mathbf{p}_i = (x_i, y_i)$, and we assume that the basis arguments (the first $c$ arguments to the hash function) are non-collinear. In both cases, the center of the coordinate system is defined as the barycenter of the basis tuple (the first $c$ vectors in the arguments to $\mathbf{h}_c$). As we saw in chapter 4, this choice maximizes the number of available symmetries, and has certain advantages in terms of avoiding compounding of errors.

Each hash function has the form

$$\mathbf{h}(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_{c+1}) = \begin{pmatrix} u \\ v \end{pmatrix} = A^{-1}\mathbf{b}$$

where the $A$ and $\mathbf{b}$ appear in Eqns. 6.7 and 6.10 respectively.[7]

It can be shown that the functions $\mathbf{h}_s$ and $\mathbf{h}_a$ are respectively similarity-specific and affine-specific. We recall from section 7.1 that this means that the fibers of the hash functions are orbits of the respective transformation classes so that if $\mathbf{h}(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_{c+1}) = \left(\mathbf{p}'_1, \mathbf{p}'_2, \ldots, \mathbf{p}'_{c+1}\right)$ then the features $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_{c+1}$ can be transformed to the features $\mathbf{p}'_1, \mathbf{p}'_2, \ldots, \mathbf{p}'_{c+1}$ by a transformation in the respective class. Thus if a hash value equals (or is very close to) a prerecorded entry then we know that there is a transformation that brings the collection of features from the model close to the set of features from the image.

The two distribution functions of an entry due to positional noise in the image point features can be derived very easily. Recall that we make the assumption of exact-matching hypotheses. Consequently in each case we assume that the first $c$ arguments are fixed and that the final point is perturbed by additive Gaussian positional noise with zero mean and covariance $\left(\begin{smallmatrix} \sigma & 0 \\ 0 & \sigma \end{smallmatrix}\right)$. Thus $\mathbf{p}_1$ $\mathbf{p}_2$ $\ldots$ $\mathbf{p}_c$ are fixed and $\mathbf{X}$ is a random vector with mean $\mathbf{p}_{c+1}$ and covariance $\left(\begin{smallmatrix} \sigma & 0 \\ 0 & \sigma \end{smallmatrix}\right)$. Let us consider the random vector $\mathbf{h}(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{X})$. Since $\mathbf{h}$ is linear in $\mathbf{p}_{c+1}$ we conclude that $\mathbf{h}(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{X})$ is also a Gaussian with mean $\mathbf{h}(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_{c+1})$ and covariance

$$\Sigma = \sigma^2 \left[A^t A\right]^{-1} .$$

More specifically for the case of similarity we have that

$$\boxed{\Sigma_s = \frac{\sigma^2}{\|\mathbf{p}_1 - \mathbf{p}_2\|^2} \cdot \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}} . \tag{7.7}$$

For the case of affine transformation we define $\boldsymbol{\delta}_1 = \mathbf{p}_2 - \mathbf{p}_1$ and $\boldsymbol{\delta}_2 = \mathbf{p}_3 - \mathbf{p}_1$.

[7] For the special case $\alpha = \beta = \gamma = 1/3$.

Then the covariance matrix is

$$\boxed{\Sigma_a = \frac{\sigma^2}{\mid \delta_{1,x}\delta_{2,y} - \delta_{2,x}\delta_{1,y} \mid^2} \cdot \begin{pmatrix} \delta_{1,y}^2 + \delta_{2,y}^2 & -\delta_{1,x}\delta_{1,y} - \delta_{2,x}\delta_{2,y} \\ -\delta_{1,x}\delta_{1,y} - \delta_{2,x}\delta_{2,y} & \delta_{1,x}^2 + \delta_{2,x}^2 \end{pmatrix}}. \quad (7.8)$$

Next we compute the distribution functions $f_{\mathbf{Y}|\mathcal{H}}(\boldsymbol{\xi})$ and $f_{\mathbf{Y}}(\boldsymbol{\xi})$. We use $\mathbf{Y} = \mathbf{h}(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{X})$ where $\mathbf{X}$ is now a random vector that takes on image features (throughout the entire image). For the unconditioned case ($f_{\mathbf{Y}}$) we assume that $\mathbf{X}$ takes on with equal probability any of $S - c$ features located uniformly throughout the image. For the conditioned case $\left(f_{\mathbf{Y}|\mathcal{H}}\right)$ then we know that there are $n - c$ positions where features are likely to occur due to the completion of the interpretation represented by the hypothesis $\mathcal{H}$. If we assume that the rate of non-obscuration of model points in the image is $\rho$ then there are $(n - c)$ known positions with total density $\rho(n - c)$ which will correspond to Gaussian spikes in the space of invariants and the remaining $S - c - \rho(n - c)$ total density of points will be evenly distributed throughout the image. Because the number of excess features $d$ is one there is no "surface" density support.

Let $\mathcal{R}$ denote the image region (usually rectangular). The support of $f_{\mathbf{Y}}(\boldsymbol{\xi})$ in the space of invariants is simply $\mathbf{h}(\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_2}, \ldots, \mathbf{p}_{\mu_c}, \mathcal{R})$ meaning the hash function with the fixed basis applied to the entire image domain as the final parameter. Since $f_{\mathbf{Y}}$ is assumed constant over its support and integrates to $S - c$ we have that

$$f_{\mathbf{Y}}(\boldsymbol{\xi}) \equiv \frac{S - c}{\mid \det(A^{-1}) \mid \cdot Area(\mathcal{R})}$$

where the function $Area(\cdot)$ returns the area of its argument.

For the conditioned function we have Gaussian spikes at each entry $\omega_j$ located

at

$$\boldsymbol{\zeta}_j = \boldsymbol{\zeta}(\omega_j) = \mathbf{h}(\mathbf{f}_{m,j_1}, \mathbf{f}_{m,j_2}, \ldots, \mathbf{f}_{m,j_c}, \mathbf{f}_{m,j}),$$

where $\omega_j$ is the entry $[m, [j_1, j_2, \ldots, j_c, j]]$ and it is understood that $m, j_1, j_2, \ldots, j_c$ are fixed and $j$ runs through the indices $1, 2, \ldots, n$ leaving out the basis indices $\{j_1, j_2, \ldots, j_c\}$. The entry is tagged with the information $[m, [j_1, j_2, \ldots, j_c]]$ which is the same information that is fixed by the hypothesis $\mathcal{H}(m, [j_1, j_2, \ldots, j_c]\, \mathcal{B})$. We thus have

$$f_{\mathbf{Y}|\mathcal{H}}(\boldsymbol{\xi}) = \frac{S - c - \rho(n-c)}{|\det(A^{-1})| \cdot Area(\mathcal{R})} + \sum_j \frac{\rho}{2\pi\sqrt{|\det(\Sigma)|}} \exp\left(-\frac{\left(\boldsymbol{\xi} - \boldsymbol{\zeta}_j\right)^t \Sigma^{-1} \left(\boldsymbol{\xi} - \boldsymbol{\zeta}_j\right)}{2}\right).$$

Note that $\mathcal{H}$ is considered fixed and the $\boldsymbol{\zeta}_j$'s depend on $\mathcal{H}$.

Next we compute the log-probability ratio for each hypothesis using Eqn. 7.4. We recall that the total support for a hypothesis is denoted by $Z(\mathcal{H})$. We will assume that each *a priori* probability $\Pr(\mathcal{H})$ is constant and will omit additive constants (such as the $\log K$ term).

Using Eqn. 7.6 and the formulas for $f_{\mathbf{Y}}$ and $f_{\mathbf{Y}|\mathcal{H}}$ we obtain after some simplifications

$$\boxed{Z(\mathcal{H}) = \sum_{k=1}^{S-c} \log\left[1 - \frac{\rho(n-c)}{S-c} + \sum_j \frac{\rho \cdot Area(\mathcal{R})}{2\pi(S-c)\sigma^2} \exp\left(-\frac{\left(\boldsymbol{\xi}_k - \boldsymbol{\zeta}_j\right)^t \Sigma^{-1} \left(\boldsymbol{\xi}_k - \boldsymbol{\zeta}_j\right)}{2}\right)\right]} \tag{7.9}$$

where $\left\{\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \ldots, \boldsymbol{\xi}_{S-c}\right\}$ are the hash locations $\Gamma$ of image point features using basis $\mathcal{B}$.

We wish to exchange the order of summations. This is possible because for a given hypothesis $\mathcal{H}$ for a fixed image point and thus a fixed hash location $\boldsymbol{\xi}_k$ at

most one model hash point $\boldsymbol{\zeta}_j$ can lie near $\boldsymbol{\xi}_k$. We have assumed that the model points are distinct and separated and thus for a fixed model/basis the resulting hashes $\boldsymbol{\zeta}_1, \boldsymbol{\zeta}_2, \ldots, \boldsymbol{\zeta}_{n-c}$ are separated so that at most one entry may lie near the given point $\boldsymbol{\xi}_j$. Consequently all but one of the exponential terms is essentially zero for any given $k$. Thus we set

$$z_{k,j}(\mathcal{H}) = \begin{cases} \log\left[1 - \frac{\rho(n-c)}{S-c} + \frac{\rho \cdot Area(\mathcal{R})}{2\pi(S-c)\cdot\sigma^2} \exp\left(-\frac{\left(\boldsymbol{\xi}_k - \boldsymbol{\zeta}_j\right)^t \Sigma^{-1} \left(\boldsymbol{\xi}_k - \boldsymbol{\zeta}_j\right)}{2}\right)\right] \\ 0 \end{cases}$$

according to whether $\|\boldsymbol{\xi}_k - \boldsymbol{\zeta}_j\|$ is small (first line) or large (whence $z_{k,j}$ is set to zero). We then have approximately

$$Z(\mathcal{H}) \approx \sum_{j=1}^{n-c} \sum_{k=1}^{S-c} z_{k,j}(\mathcal{H}) \ .$$

Other definitions for $z_{k,j}$ are possible and more elegant but there is a potential advantage in assigning zero values for many of the variables.

Finally we describe the Bayesian geometric hashing algorithm for each of the two transformations. The space of invariants contains entries with model/basis tags of the form $[m, [j_1, j_2, \ldots, j_c]]$. The entries are located at positions $\boldsymbol{\zeta} = \mathbf{h}(\mathbf{f}_{m,j_1}, \mathbf{f}_{m,j_2}, \ldots, \mathbf{f}_{m,j_c}, \mathbf{f}_{m,j})$; $\mathbf{f}_{m,j}$ is an arbitrary point in the model not included in the first $c$ arguments (the basis tuple). During the recognition phase we chose a basis of point features $\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_2}, \ldots, \mathbf{p}_{\mu_c}$ and perform a probe during which we will apply votes to entries. Initially a vote of zero is stored for each entry. We compute hash locations $\boldsymbol{\xi} = \mathbf{h}(\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_2}, \ldots, \mathbf{p}_{\mu_c}, \mathbf{p}_k)$ for point features $\mathbf{p}_k \in \mathcal{S}'$ i.e. using the remaining point features in the image. For each such hash $\boldsymbol{\xi}$ we locate all entries at nearby positions $\boldsymbol{\zeta}$. For each such entry we compute the value $w(\boldsymbol{\zeta}, \boldsymbol{\xi})$ for the appropriate $c$. For similarity invariance $(c = 2)$ we have

$$\boxed{w_s(\boldsymbol{\zeta},\boldsymbol{\xi}) = \log\left[1 - \frac{\rho(n-2)}{S-2} + \frac{\rho \cdot Area(\mathcal{R})}{2\pi(S-2)\sigma^2}\exp\left(\frac{-\|\boldsymbol{\xi}-\boldsymbol{\zeta}\|^2}{2\sigma^2/\|\mathbf{p}_{\mu_1}-\mathbf{p}_{\mu_2}\|^2}\right)\right]}\ .$$

For affine invariance $(c=3)$ we set $\boldsymbol{\delta}_1 = \mathbf{p}_{\mu_2} - \mathbf{p}_{\mu_1}$ and $\boldsymbol{\delta}_2 = \mathbf{p}_{\mu_3} - \mathbf{p}_{\mu_1}$. Then

$$\boxed{\begin{aligned} w_a(\boldsymbol{\zeta},\boldsymbol{\xi}) = \log\Bigg[1 &- \frac{\rho(n-3)}{S-3} \\ &+ \frac{\rho \cdot Area(\mathcal{R})}{2\pi(S-3)\sigma^2}\exp\left(\frac{-\|(\boldsymbol{\xi}_x-\boldsymbol{\zeta}_x)\cdot\boldsymbol{\delta}_1 + (\boldsymbol{\xi}_y-\boldsymbol{\zeta}_y)\cdot\boldsymbol{\delta}_2\|^2}{2\sigma^2}\right)\Bigg] \end{aligned}}\ .$$

We might note that in both cases the argument to the exponential function is one-half the square of the distance measured relative to the standard deviation $\sigma$ between the predicted hash location of the corresponding point in the image and the observed location of the point.

Having computed the weight $w(\boldsymbol{\zeta},\boldsymbol{\xi})$ for the entry $\omega$ we update the $z$ value at $\omega$ by replacing $z(\omega)$ with the maximum of the current value and the weight. This is done for every entry $\omega$ in the neighborhood of the hash $\boldsymbol{\xi}$ and is repeated for every hash from the scene. The probe is concluded by summing the $z$ values associated with entries having equal tags:

$$Z(\mathcal{H}) = \sum_j z([m,[j_1,j_2,\ldots,j_c,j]])\ .$$

Here $\mathcal{H} = [m,[j_1,j_2,\ldots,j_c]]$. The hypotheses with the top few $Z(\mathcal{H})$ values are candidates for interpretations using the basis $\mathcal{B}$.

Since the maximum value that a hash can contribute to a hypothesis is

$$z_0 = \log\left(1 - \frac{\rho(n-c)}{S-c} + \frac{\rho \cdot Area(\mathcal{R})}{2\pi(S-c)\sigma^2}\right),$$

we know that if a hypothesis with a total vote of $Z(\mathcal{H})$ must have at least $Z(\mathcal{H})/z_0$ corroborating points in the neighborhood of the positions predicted

by the hypothesized match. A summary of the $2D$ point pattern recognition algorithm is given in Figure 7.6.

Finally we briefly consider the size of the neighborhood in which nearby hashes should be accessed. Given a hash at location $\boldsymbol{\xi}$ the vote applied to nearby entries is $w(\boldsymbol{\xi}, \boldsymbol{\zeta})$ and zero for entries that are far away (the vote is applied in a maximum

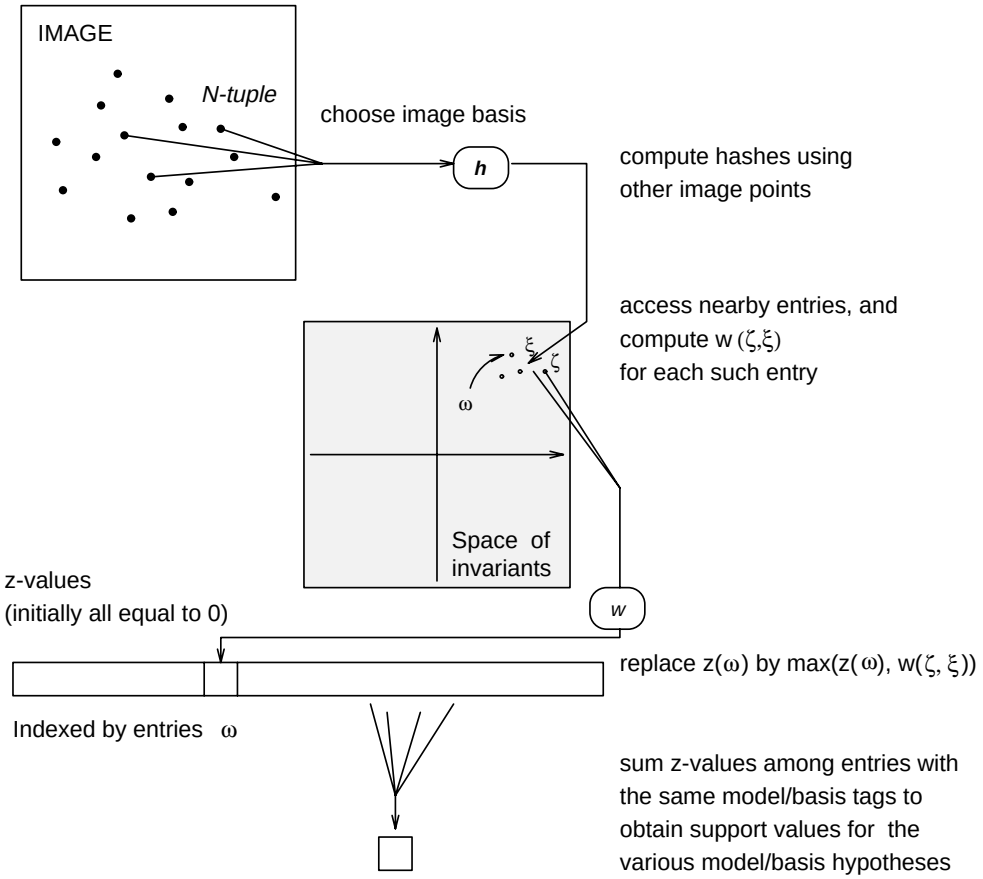


**Figure 7.6 The steps for a probe with a single basis set during the recognition phase of the Bayesian geometric hashing algorithm for point pattern recognition.**

way). Since the function $w(\boldsymbol{\xi}, \boldsymbol{\zeta})$ decays rapidly as the separation increases the neighborhood can be as large as we like. If a vote is applied to two entries with

the same tag, then the approximation to Eqn. 7.9 is violated, but due to the separation of hashes with equal tags, at least one of those votes will be close to zero, which has no consequence to the vote at that site. However, it is desirable to use a small neighborhood, to reduce the number of entries to which votes must be applied. It suffices to find a distance at which the weight is guaranteed to be small, i.e. a small fraction of its maximum value $z_0$, relative to the $n - c$ entries that will be combined to form the total vote $Z(\mathcal{H})$.

## 7.10 The Formulas: Approximate Matching

In this section we present the formulas for the case of weighted voting and approximate-matching hypotheses.

We modify the hypotheses so that each hypothesis represents an approximate-matching between the selected basis and a model/basis combination. $\mathcal{H}$ will now imply an approximate matching. As in the exact matching case, two classes of transformations will be considered, namely similarity and affine.

The hash functions again operate on a basis set plus one more feature and have the form

$$\mathbf{h}(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_{c+1}) = \begin{pmatrix} u \\ \\ v \end{pmatrix} = A^{-1}\mathbf{b}$$

where the $A$ and $\mathbf{b}$ appear in Eqns. 6.7 and 6.10 respectively.[8]

Since we make the assumption of approximate mathing hypotheses, such an analysis must take into account potential noise in the basis features $\mathcal{B}$. And this was the study in chapter 6, where we examined how the error propagates if we assume that the positions of all the point features of the $(c+1)$-tuple are

[8] For the special case $\alpha = \beta = \gamma = 1/3$.

disturbed by additive Gaussian noise. The conclusion of that analysis was that $\mathbf{h}(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{X})$ is also a Gaussian with mean $\mathbf{h}(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_{c+1})$ and covariance matrices

$$\Sigma_s = \frac{(4 \parallel (u,v) \parallel^2 +3)\,\sigma^2}{2 \parallel \mathbf{p}_2 - \mathbf{p}_1 \parallel^2} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} ,$$

and

$$\Sigma_a = \frac{(4(u^2+v^2+uv)+8/3)\,\sigma^2}{2 \mid (\mathbf{p}_2 - \mathbf{p}_1)(\mathbf{p}_3 - \mathbf{p}_1)^{\perp^t} \mid^2} \begin{pmatrix} \parallel (\mathbf{p}_3 - \mathbf{p}_1) \parallel^2 & -(\mathbf{p}_2 - \mathbf{p}_1)(\mathbf{p}_3 - \mathbf{p}_1)^t \\ -(\mathbf{p}_2 - \mathbf{p}_1)(\mathbf{p}_3 - \mathbf{p}_1)^t & \parallel (\mathbf{p}_2 - \mathbf{p}_1) \parallel^2 \end{pmatrix}$$

for the similarity and affine cases respectively.

We next compute the distribution functions $f_{\mathbf{Y}|\mathcal{H}}(\boldsymbol{\xi})$ and $f_{\mathbf{Y}}(\boldsymbol{\xi})$. We use $\mathbf{Y} = \mathbf{h}(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{X})$ where $\mathbf{X}$ is now a random vector that takes on image features (throughout the entire image). As before for the unconditioned case ($f_{\mathbf{Y}}$) we assume that $\mathbf{X}$ takes on with equal probability any of $S - c$ features located uniformly throughout the image. For the conditioned case $\left(f_{\mathbf{Y}|\mathcal{H}}\right)$ we know that there are $n - c$ positions where features are likely to occur due to the completion of the interpretation represented by the hypothesis $\mathcal{H}$. If $\rho$ denotes the rate of non-obscuration of model point features in the image then there are $(n - c)$ known positions with total density $\rho(n - c)$ which will correspond to Gaussian spikes in the space of invariants and the remaining $S - c - \rho(n - c)$ total density of points will be evenly distributed throughout the image. Because the number of excess features $d$ is one there is no "surface" density support.

Unlike the exact-matching case we use the *same* probability density function for the invariants independent of the selected basis $\mathcal{B}$. In chapter 4 we determined analytically the expected probability density function $f_e(\boldsymbol{\xi})$ for invariants over all possible basis selections for a number of transformation and

feature distribution combinations. Since $f_{\mathbf{Y}}$ integrates to $S-c$ we have that

$$f_{\mathbf{Y}}(\boldsymbol{\xi}) \equiv (S-c) \cdot f_e(\boldsymbol{\xi})$$

Alternatively and depending on the application $f_e(\boldsymbol{\xi})$ and thus $f_{\mathbf{Y}}$ can be determined empirically. For the conditioned function we have Gaussian spikes at each entry $\omega_j$ located at

$$\boldsymbol{\zeta}_j = \boldsymbol{\zeta}(\omega_j) = \mathbf{h}(\mathbf{f}_{m,j_1}, \mathbf{f}_{m,j_2}, \ldots, \mathbf{f}_{m,j_c}, \mathbf{f}_{m,j}) \ ,$$

where $\omega_j$ is the entry $[m,[j_1,j_2,\ldots,j_c,j]]$ and it is understood that $m,j_1,j_2,\ldots,j_c$ are fixed and $j$ runs through the indices $1,2,\ldots,n$ leaving out the basis indices $\{j_1,j_2,\ldots,j_c\}$. The entry is tagged with the information $[m,[j_1,j_2,\ldots,j_c]]$ which is the same information that is fixed by the hypothesis $\mathcal{H}(m,[j_1,j_2,\ldots,j_c]\,\mathcal{B})$. We thus have

$$\begin{aligned} f_{\mathbf{Y}|\mathcal{H}}(\boldsymbol{\xi}) = \ & (S-c-\rho(n-c)) \cdot f_e(\boldsymbol{\xi}) \\ & + \sum_j \frac{\rho}{2\pi\sqrt{|\det(\Sigma_j)|}} \exp\left(-\frac{\left(\boldsymbol{\xi}-\boldsymbol{\zeta}_j\right)\Sigma_j^{-1}\left(\boldsymbol{\xi}-\boldsymbol{\zeta}_j\right)}{2}\right) \ . \end{aligned}$$

Note that $\mathcal{H}$ is considered fixed and that *both* $\Sigma_j$ and $\boldsymbol{\zeta}_j$ depend on $\mathcal{H}$.

We can now compute the log-probability ratio for each approximate hypothesis using Eqn. 7.4. Once again $Z(\mathcal{H})$ denotes the total support for a hypothesis each $\Pr(\mathcal{H})$ is constant and additive constants (i.e. the $\log K$ term) are omitted. Using Eqn. 7.6 and the formulas for $f_{\mathbf{Y}}$ and $f_{\mathbf{Y}|\mathcal{H}}$ we obtain

$$\boxed{\begin{aligned} Z(\mathcal{H}) = \sum_{k=1}^{S-c} \log \Bigg[ & 1 - \frac{\rho(n-c)}{S-c} \\ & + \sum_j \frac{\rho(S-c)^{-1}\left[f_e(\boldsymbol{\xi})\right]^{-1}}{2\pi\sqrt{|\det(\Sigma_j)|}} \exp\left(-\frac{\left(\boldsymbol{\xi}_k-\boldsymbol{\zeta}_j\right)\Sigma_j^{-1}\left(\boldsymbol{\xi}_k-\boldsymbol{\zeta}_j\right)}{2}\right)\Bigg] \end{aligned}} \tag{7.10}$$

where $\left\{\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \ldots, \boldsymbol{\xi}_{S-c}\right\}$ are the hash locations $\Gamma$ of image point features using basis $\mathcal{B}$.

Working as in section 7.9 we can show that

$$z_{k,j}(\mathcal{H}) = \begin{cases} \log\left[1 - \frac{\rho(n-c)}{S-c} + \frac{\rho(S-c)^{-1}\left[f_e(\boldsymbol{\xi})\right]^{-1}}{2\pi\sqrt{|\det(\Sigma_j)|}} \exp\left(-\frac{\left(\boldsymbol{\xi}_k - \boldsymbol{\zeta}_j\right)\Sigma_j^{-1}\left(\boldsymbol{\xi}_k - \boldsymbol{\zeta}_j\right)}{2}\right)\right] \\ 0 \end{cases}$$

according to whether $\left(\boldsymbol{\xi}_k - \boldsymbol{\zeta}_j\right)\Sigma_j^{-1}\left(\boldsymbol{\xi}_k - \boldsymbol{\zeta}_j\right)$ is small (first line) or large (whence $z_{k,j}$ is set to zero). Approximately we can write

$$Z(\mathcal{H}) \approx \sum_{j=1}^{n-c}\sum_{k=1}^{S-c} z_{k,j}(\mathcal{H}) \ .$$

The Bayesian geometric hashing algorithm for each of the two transformations and the assumption of approximate hypotheses proceeds as before: the space of invariants contains entries with model/basis tags of the form $[m, [j_1, j_2, \ldots, j_c]]$ located at positions $\boldsymbol{\zeta} = \mathbf{h}(\mathbf{f}_{m,j_1}, \mathbf{f}_{m,j_2}, \ldots, \mathbf{f}_{m,j_c}, \mathbf{f}_{m,j})$ where $\mathbf{f}_{m,j}$ is an arbitrary point in the model not included in the first $c$ arguments (the basis tuple). During the recognition phase we chose a basis of image point features $\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_2}, \ldots, \mathbf{p}_{\mu_c}$ and perform a probe during which we will apply votes to entries. Initially a vote of zero is stored for each such entry. We compute hash locations $\boldsymbol{\xi} = \mathbf{h}(\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_2}, \ldots, \mathbf{p}_{\mu_c}, \mathbf{p}_k)$ for point features $\mathbf{p}_k \in \mathcal{S}'$ i.e. using the remaining point features in the image. For each such hash $\boldsymbol{\xi}$ we locate all entries at nearby positions $\boldsymbol{\zeta}$. For each such position we compute the value $w(\boldsymbol{\zeta}, \boldsymbol{\xi})$ for the appropriate $c$. Simple substitution shows that for similarity invariance ($c = 2$) we have

$$w_s(\boldsymbol{\zeta},\boldsymbol{\xi}) \;=\; \log\left[1 - \frac{\rho(n-2)}{S-2} + \frac{\rho\|\mathbf{p}_{\mu_2} - \mathbf{p}_{\mu_1}\|^2}{\pi\sigma^2\left(4\|\boldsymbol{\zeta}\|^2+3\right)(S-2)\,f_e(\boldsymbol{\xi})} \cdot \exp\left(\frac{-\|\boldsymbol{\xi}-\boldsymbol{\zeta}\|^2}{\sigma^2\left(4\|\boldsymbol{\zeta}\|^2+3\right)/\|\mathbf{p}_{\mu_2} - \mathbf{p}_{\mu_1}\|^2}\right)\right]$$

For affine invariance $(c=3)$ we set $\boldsymbol{\delta}_1 = \mathbf{p}_{\mu_2} - \mathbf{p}_{\mu_1}$ and $\boldsymbol{\delta}_2 = \mathbf{p}_{\mu_3} - \mathbf{p}_{\mu_1}$. Then

$$w_a(\boldsymbol{\zeta},\boldsymbol{\xi}) \;=\; \log\left[1 - \frac{\rho(n-3)}{S-3} + \frac{\rho\,|\,(\mathbf{p}_{\mu_2} - \mathbf{p}_{\mu_1})(\mathbf{p}_{\mu_3} - \mathbf{p}_{\mu_1})^{\perp^t}\,|^2}{\pi\sigma^2\left(4\left(\|\boldsymbol{\zeta}\|^2 + \boldsymbol{\zeta}_x\boldsymbol{\zeta}_y\right)+8/3\right)(S-3)\,f_e(\boldsymbol{\xi})} \cdot \exp\left(\frac{-\|\,(\boldsymbol{\xi}_x - \boldsymbol{\zeta}_x)\cdot\boldsymbol{\delta}_1 + \left(\boldsymbol{\xi}_y - \boldsymbol{\zeta}_y\right)\cdot\boldsymbol{\delta}_2\|^2}{\sigma^2\left(4\left(\|\boldsymbol{\zeta}\|^2 + \boldsymbol{\zeta}_x\boldsymbol{\zeta}_y\right)+8/3\right)/\,|\,(\mathbf{p}_{\mu_2} - \mathbf{p}_{\mu_1})(\mathbf{p}_{\mu_3} - \mathbf{p}_{\mu_1})^{\perp^t}\,|^2}\right)\right]$$

In the above two formulas $f_e(\boldsymbol{\xi})$ should be substituted by the appropriate expression for the probability density; the expression can either be taken from the analysis of chapter 4 or derived empirically.

Having computed the weight $w(\boldsymbol{\zeta},\boldsymbol{\xi})$ for the entry $\omega$ we update the $z$ value at $\omega$ by replacing $z(\omega)$ with the maximum of the current value and the weight. This is done for every entry $\omega$ in the neighborhood of the hash $\boldsymbol{\xi}$ and is repeated for every hash from the scene. The probe is concluded by summing the $z$ values associated with with entries with equal tags:

$$Z(\mathcal{H}) = \sum_j z([m,[j_1,j_2,\ldots,j_c,j]]) \;\;.$$

Here $\mathcal{H} = [m,[j_1,j_2,\ldots,j_c]]$. The hypotheses with the top few $Z(\mathcal{H})$ values are candidates for interpretations using the basis $\mathcal{B}$. Figure 7.6 shows a summary of

the algorithm: the approximate-hypotheses expressions for $w(\cdot,\cdot)$ and $z(\cdot)$ are to be used.

We should mention again that our use of the approximate-matching hypotheses implies that in the Bayesian analysis above the evidence is no longer independent. In fact the error in the basis features can be deduced from a sufficient number of corroborating hashes involving features from the model that is embedded in the scene. As a result of the lack of independence the values we compute give only an approximate measure of the relative *a posteriori* probabilities.

We already stated that experimental evidence with both synthetic and real-world data supports the use of approximate-matching hypotheses; in the next chapter we will present experimental results from the implementation of a system that makes use of the approximate-matching approach to the accumulation of evidence.

# Chapter 8

# Experimental Results

In this chapter we demonstrate the validity of the Bayesian geometric hashing algorithm that uses the approximate matching hypotheses (see section 7.10). In particular, we describe in detail the actual implementation of a complete object recognition system.

The recognition system is implemented on a $8K$-processor $CM$-2 and can recognize objects that have undergone a similarity transformation (i.e. rotation, translation, and scaling), from a library of 32 models. The models we use are military aircraft and production automobiles.

We test the approach using real-world imagery; in particular, the test inputs are photographic data of military aircraft in flight, and of automobiles in street scenes. The resulting system is scalable, works rapidly and very efficiently on an $8K$-processor machine, and the quality of results is excellent.

## 8.1 Off-Line Preprocessing

Since our intention was to build a complete object recognition system, we incorporated an automatic feature extraction mechanism. A straightforward boundary

following algorithm[1] was applied to the output of the edge detection stage, followed by a simple divide-and-conquer polygonal approximation algorithm [27]. The edge detector that we used is the one described by Cox and Boie in [13].[2] The output of the edge detection stage was a collection of curves (an *edge map*), and polygonal approximations for each of those curves were determined; curves shorter than 100 pixels were not considered. No other filtering or preprocessing was performed.

There was no attempt to implement the edge detection, boundary following and feature extraction stages in parallel. Indeed, the corresponding code is serial and runs on the Front End (see also [81]). Descriptions of very fast parallel algorithms for these operations, based on replicating data structures, can be found in [71,72].

The database in our experiments contained thirty-two models: fourteen of the models were military aircrafts, whereas the remaining eighteen were automobiles (six automobiles seen from three different viewpoints). The database models were allowed to undergo similarity transformations (i.e. rotation, translation and scaling).

For the aircraft models, the profile drawings of fourteen military aircraft from [77] were scanned using a Microtek 300 color/grayscale scanner. These drawings are not photographs: they are schematic drawings that are probably drawn roughly to scale. The scanner is capable of a resolution of 300 dpi, however, we used 120 dpi resolution to digitize the drawings.

For the automobile models, we obtained photographs of six different automo-

[1] The boundary following algorithm assumes eight-connectivity.
[2] The value of the filter's $\sigma$ was typically equal to 1.5 (see [13]).

biles seen from three different viewpoints: the camera was at the height of the automobiles' midline, with its optical axis pointing at the middle of the automobiles' long side. The three viewpoints corresponded to azimuth values of roughly $-45$, 0 and $+45$ degrees. The average distance from the automobiles was approximately fifty feet. The photographs were subsequently scanned at a resolution of 75 dpi. The fourteen aircraft and six automobile types contained in our database, appear in table 8.1.

| | |
|---|---|
| A-4 *Skyhawk* | A-6 *Intruder* |
| A-10 *Thunderbolt* | F-14 *Tomcat* |
| F-15 *Eagle* | F-16 *Falcon* |
| F/A-18 *Hornet* | Mig-21 *Fishbed* |
| Mig-23 *Flogger* | Mig-29 *Fulcrum* |
| Mig-31 *Foxhound* | Mirage 2000 |
| Sea Harrier | Panavia Tornado |
| Chevrolet *Astro* (lateral) | Chevrolet *Astro* (oblique frontal) |
| Chevrolet *Astro* (oblique rear) | Dodge *Dart* (lateral) |
| Dodge *Dart* (oblique frontal) | Dodge *Dart* (oblique rear) |
| Ford *Econoline150* (lateral) | Ford *Econoline150* (oblique frontal) |
| Ford *Econoline150* (oblique rear) | Chrysler *Horizon* (lateral) |
| Chrysler *Horizon* (oblique frontal) | Chrysler *Horizon* (oblique rear) |
| Honda *Civic* (lateral) | Honda *Civic* (oblique frontal) |
| Honda *Civic* (oblique rear) | Volvo *S.-W.* (lateral) |
| Volvo *S.-W.* (oblique frontal) | Volvo *S.-W.* (oblique rear) |

**Table 8.1 The thirty-two models of the database.**

The vertices of the different approximating polygons coincided with either points of discontinuity in the tangent direction of the model's contour (i.e., vertices or points of very high curvature), or points of maximum curvature. A subset of sixteen points was selected from each model's point feature set. It should be stressed that this model feature selection is carried out during the building of the

database and thus performed off-line. Figure 8.1 shows the edge maps and the selected points for three of the database models: the F-16 *Falcon*, the Sea Harrier and the Ford *Econoline150*.

Clearly, more sophisticated approaches, such as spline fitting, could be used to determine the feature set: our choice for the feature detection mechanism reflected our desire to determine the limitations of the proposed object recognition system.

## 8.2 The Two-level Randomized Algorithm

In this section, we describe in more detail the probe selection mechanism, which is independent of whether we use the formulas for approximate or exact matching.

The Bayesian geometric hashing algorithm necessitates that a basis probe with $c$ members be selected from the point feature set. Given such a probe, hashes are determined for all of the remaining $S - c$ image point features and $z$-values are computed (see Figure 7.6).

The main component of our probe selection algorithm takes a straightforward approach: the basis members are selected without replacement and uniformly from the point feature set.[3] This randomized algorithm is both simple and efficient. Indeed, if $S$ is the number of image features, $n$ the number of model features, and $\theta n$ the number of unoccluded model points, the probability that $s$ selections will be needed before we encounter a basis probe consisting only of model features is given by the geometric distribution $d_g(\cdot, \cdot)$

$$d_g\left(s,\ \prod_{i=1}^{c}\left(\frac{\theta\ n-i+1}{S-i+1}\right)\right) = \prod_{i=1}^{c}\left(\frac{\theta\ n-i+1}{S-i+1}\right)\cdot\left(1-\prod_{i=1}^{c}\left(\frac{\theta\ n-i+1}{S-i+1}\right)\right)^{s-1}\ .$$

[3]However, in our current implementation of the algorithm, the different bases are selected *with* replacement, i.e. a given basis may be selected more than once before recognition occurs.

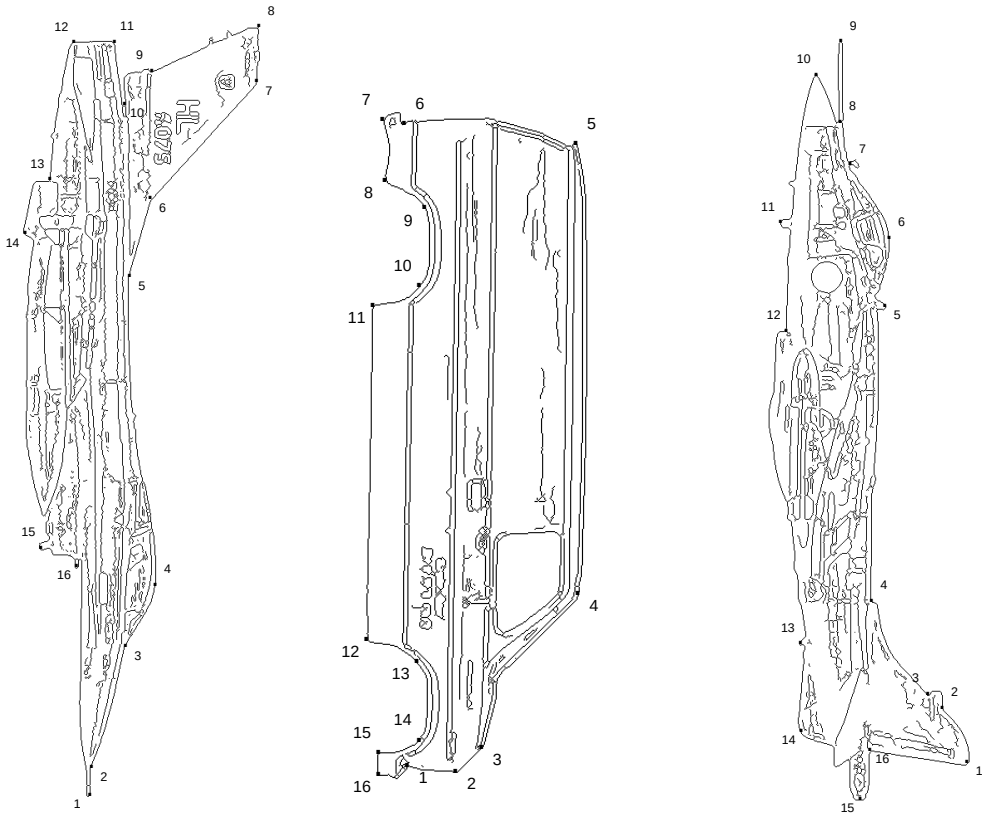


**Figure 8.1 The edge maps and the selected feature points for the database models of the F-16 *Falcon*, the Ford *Econoline150*, and the Sea Harrier.**

For $\theta = 1$, $n = 16$, $S = 150$, and $c = 2$ we can see that a probe consisting entirely of model features will be encountered with probability 0.99 after about 30 probes have been considered. This is very useful since the required number of probes represent roughly 0.1% of the $22,350$ possible two-member probes.

Clearly, the algorithm will frequently select either a poor basis, or a basis where at least one of the $c$ members does not belong to the model that is embedded in the test image. In both cases, the probe will be discarded after evidence from all $S - c$ hashes has been accumulated. However, it is possible to discard such a probe after having considered only a *fraction* of the image features.

Let us assume that a given probe consists *entirely* of model features. Let us further assume that we only consider a fraction $k$ of the remaining image features $S-c$ by uniformly selecting without replacement among them: in other words, we determine hashes and $z$-values for only $(S-c)/k$ image features. The probability that *precisely* $l$ features, with $l \leq (\theta n) - c$, out of the selected $(S-c)/k$ belong to the embedded model is given by the hypergeometric distribution $d_h(\cdot,\cdot,\cdot)$

$$d_h\left(\frac{S-c}{k}, l, (\theta n) - c\right) = \frac{\begin{pmatrix} (\theta n) - c \\ l \end{pmatrix} \begin{pmatrix} S - (\theta n) \\ \frac{S-c}{k} - l \end{pmatrix}}{\begin{pmatrix} S - c \\ \frac{S-c}{k} \end{pmatrix}} .$$

Then, the probability that *at least* $l$ points of the embedded model belong to the selected fraction of image features is

$$\boxed{\mathbb{P}_l = \sum_{j=l}^{(\theta n)-c} d_h\left(\frac{S-c}{k}, j, (\theta n) - c\right)} \tag{8.1}$$

If we know the average contribution $z_{av}$ of a hash, and if $max\{Z(\cdot,\cdot)\} < l \cdot z_{av}$, then we can discard the current probe at this point, and select another one. If on

the other hand $max\{Z(\cdot,\cdot)\} \geq l \cdot z_{av}$, we proceed and accumulate evidence from the remaining $(k-1)(S-c)/k$ image features as well.

One can think of the $1/\mathbb{P}_l$ value as the *expected slowdown* that results from the fact that only a fraction of image features is used. In order for the modified probing algorithm to be useful, the *effective speedup* must be

$$\boxed{k \cdot \mathbb{P}_l > 1}\ .$$

The value of this last expression is a function of $k$, $l$, $S$, $\theta$ and $n$. In Figure 8.2 we show some of the contours of the expression for $\theta = 1$, $n = 16$, $S = 150$ and $c = 2$, as a function of $l$ and $k$. The contour labels correspond to the value of the effective speedup. We can see that the maximum effective speedup occurs when we consider only one-seventh of the image features. But when only a small fraction of the image features is considered, the probability of a large number of model features occurring in the selected set of features is very small. Other combinations of $k$ and $l$ are more preferable: in particular, if we require that at least 5 model features occur in the selected set of features, theoretically we can achieve an expected speedup of 1.8, and we do not need to consider more than one third of the entire set of features. During the recognition process, and after $(S-2)/3$ features have been considered, we examine the largest value $Z(\cdot,\cdot)$: if the value is less than 5 times the average contribution $z_{av}$ that a hash contributes to any hypothesis, we discard the current probe, and select another one; otherwise, we accumulate evidence from the remaining $2(S-2)/3$ image features and proceed as usual. We have incorporated this two-level randomized algorithm into our recognition system, with $k = 3$ and $l = 5$: in all cases where $\theta = 1$, we observed a speedup roughly equal to 1.7 over the straightforward

algorithm which accumulates evidence from all the image features.

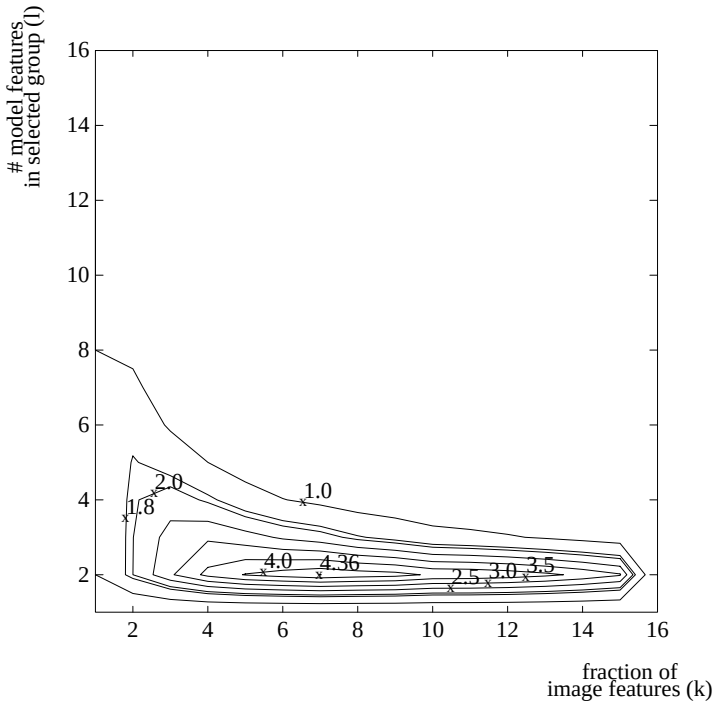


**Figure 8.2 Several of the contours for the effective speedup function. The horizontal axis corresponds to the fraction of the image features that is considered by the probe selection algorithm. The vertical axis corresponds to the least number of model features that one expects to see in the selected subset. The different contours correspond to the values of the effective speedups, and were taken at heights 1.0, 1.8, 2.0, 2.5, 3.0, 3.5, 4.0 and 4.36 respectively.**

Another helpful heuristic which we incorporated in the probing algorithm is the following: a basis tuple may be discarded if the length of any of the basis components is too small or too big; the cut-off values for our implementation were 150 and 550 pixels respectively. Further experimentation indicated that for scenes of approximately 100 points the resulting algorithm requires approximately

60 probe selections before recognition occurs.

## 8.3 Results

In this section, we describe the experimental results for our implementation of the Bayesian geometric hashing algorithm described in Figure 7.6. The expressions used for $w(\cdot,\cdot)$ and $z(\cdot)$ are the ones corresponding to the approximate matching approach (see section 7.10). The database models were allowed to undergo similarity transformations (i.e. rotation, translation and scaling). In the formulas that we used, $f_e(\cdot)$ corresponds to the case of Gaussian distributed point features (see Eqn. 4.4). The probe selection algorithm used is the two-level randomized algorithm that was described in section 8.2. All of our experiments were run on an $8K$-processor Connection Machine.

In order to test the aircraft models, we selected a number of photographs of the same aircraft type as our models, but from a different source [94]. The photographs were chosen on the basis of being taken from approximately the same viewpoint as the drawings in the model database. That is, since the model drawings are side views, and since our implementation uses only similarity invariance, recognition will only be possible with views taken generally from the side. Notably, finding such photographs is not easy, since the pictures must be taken by chase-planes. However, we emphasize that the test images are real photographs, and not drawings nor simulated data. Nor are the models taken from the same source as the photographs. The only thing that the test images and the model database have in common, other than the approximately similar viewpoints, is the aircraft types.

To test the automobile models, we obtained additional photographs of auto-

mobiles. The automobiles in our test photographs were from various locations in New York City. The only thing that the automobiles in the test photographs and those used to build our database have in common, other than the approximately similar viewpoint, is the automobile type.

All the test photographs were digitized using an *uncalibrated* CCD camera. The result was that distortions and warpings were introduced, not only from the perspective projection of the 3-D plane onto the photograph, but also from the digitization process. However, such distortions might be typical of a working vision system, and all such distortions are approximable by a similarity transformation. Edges were extracted from the resulting gray-level images, using the same edge detector that was also used during the building of the database. Again, no preprocessing or other filtering of the test images was performed. A polygonal approximation of the different edge map curves provided the points of the feature set. Figures 8.3 through 8.8 show the digitized photographs for three of our test inputs together with the corresponding edge maps and the extracted point features.

In Figure 8.3, we can see that the original photograph of the F-16 was taken with the camera positioned below the airplane's midline and towards the back of the aircraft. Further, the airplane is banking to the left. Also, notice that the F-16 in the test photograph is a two-seat trainer, unlike the model contained in our database.

The original photograph of the Sea Harrier (Figure 8.5) was very small; a juxtaposed pencil helps estimate the actual size of the original. The original picture was taken with the camera positioned in the front of the aircraft, as evidenced by the visible interior of the left engine intake. The airplane appearing

at the bottom of the photograph is a Hunter T-8M.

The photograph of the Ford *Econoline150* was taken from the *driver's* side of the automobile, unlike that of the model which was taken from the *passenger's* side. Instead of augmenting our database with entries corresponding to that viewpoint, we decided to reverse the test input and use its reflection around the vertical axis as a test input; this explains why the lettering on the front door of the vehicle appears reversed. Note that the current recognition system is not invariant to left/right reflection.

In Figures 8.9 through 8.13 we show the output of our system's implementation, for three test inputs. The retrieved database model appropriately scaled, rotated, and translated is shown overlaid on the test input. In the bottom half of each screendump, the nine top-retrieved database models are shown in order of decreasing accumulated evidence (column-major order). For each of the nine models, its name, and the retrieved basis are also indicated. The point features corresponding to each basis are marked along the contour of the corresponding model. Above each model, bars providing a length encoding of the evidence that the indicated model/basis combination has accumulated, are also shown. It should be noted here that the recovery of the transformation was based solely on the basis pair, and *not* on a best least-squares match between all the corresponding model and scene feature pairs.

As stated in these figures, approximately 24.0 milliseconds are required per probe per scene point. This is a consequence of the less than optimal use of the floating point hardware in the $CM$-2 model (hypercube model of computation).[4]

[4]Considerable speedup is possible with the $SPRINT$-chip model of computation, at the expense of requiring 64-bit floating point hardware.

**Figure 8.3 A test image for the recognition algorithm: the photograph of an F-16.**

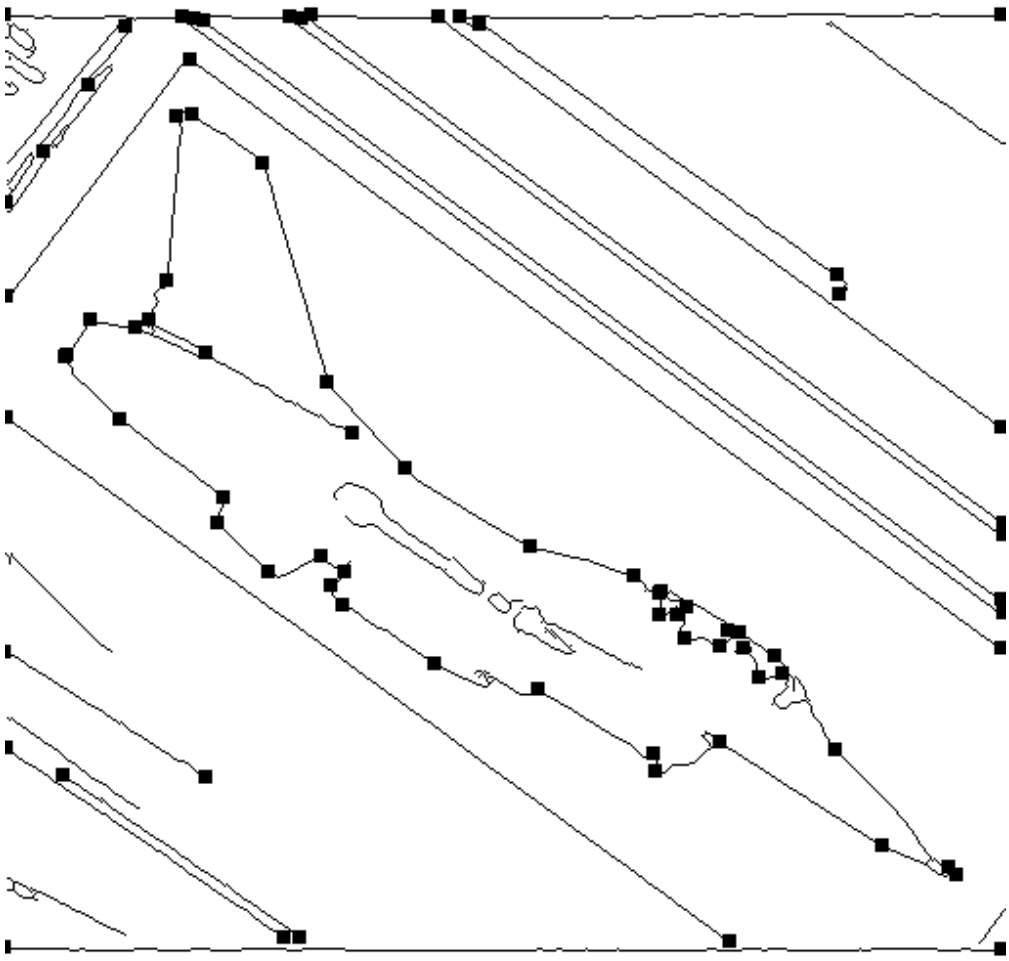

**Figure 8.4 The edge map extracted by the Cox-Boie edge detector (the value of $\sigma$ was 2.0) for the F-16 test image. Also shown are the 80 automatically extracted features.**

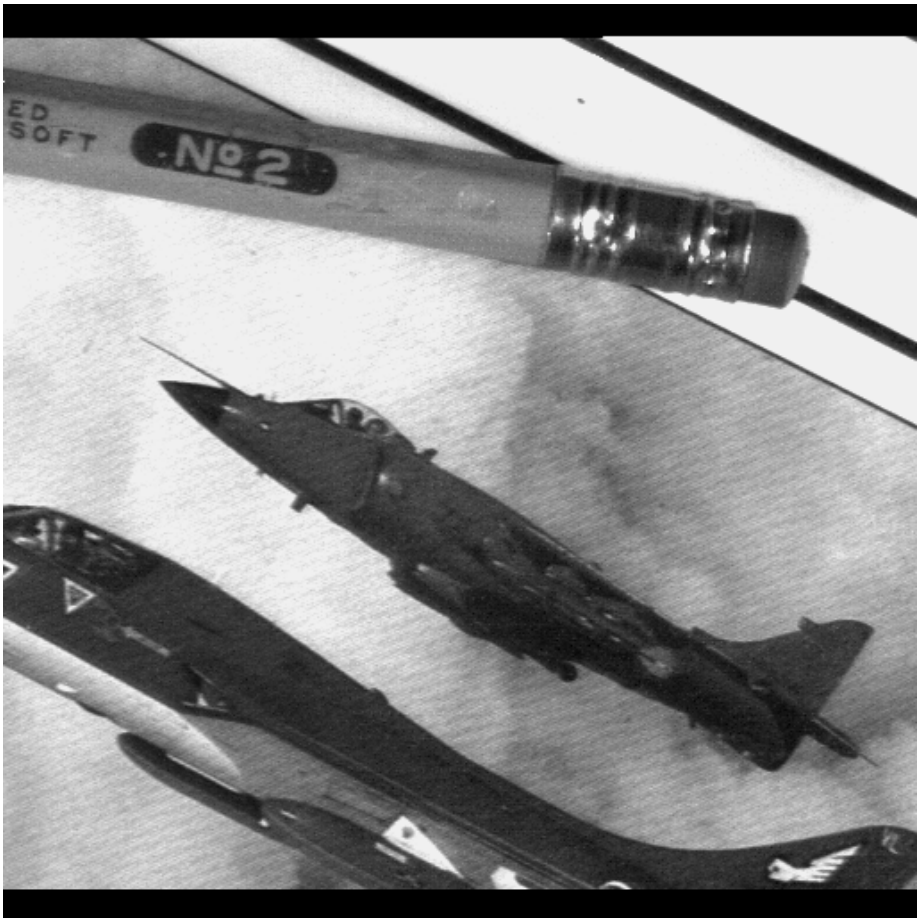


**Figure 8.5 Another test image: the photograph of a Sea Harrier. The airplane at the bottom of the picture is a Hunter T-8M.**

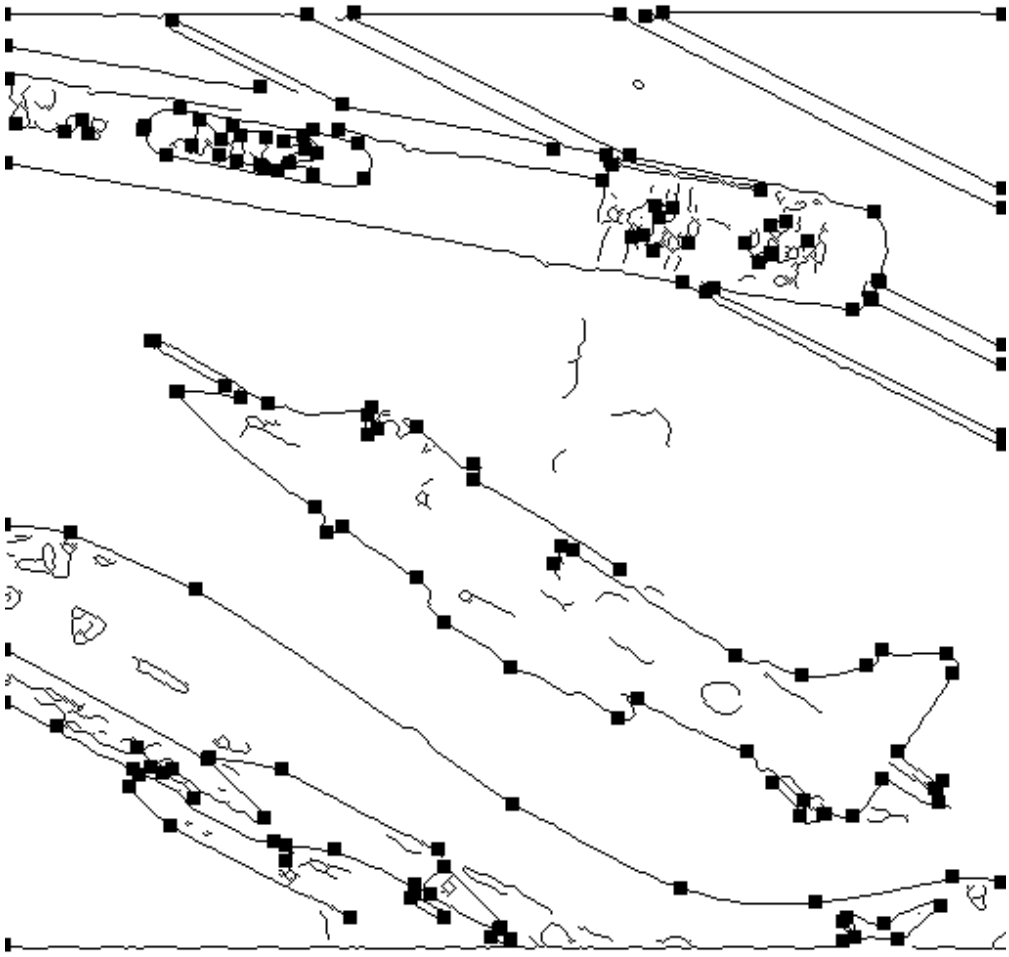

**Figure 8.6 The edge map extracted by the Cox-Boie edge detector (the value of $\sigma$ was 2.0) for the Sea Harrier test image. Also shown are the 169 automatically extracted features.**

**Figure 8.7 The test image of a Ford *Econoline150*.**

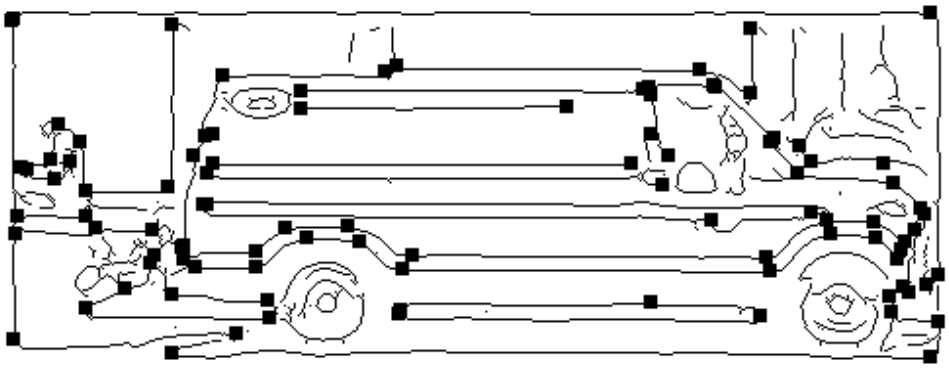

**Figure 8.8 The edge map extracted by the Cox-Boie edge detector (the value of $\sigma$ was 3.4) for the Ford *Econoline150* test image. Also shown are the 98 automatically extracted features.**

Up to thirty-two models can be included in the database, without incurring any additional requirements in processing time. As a rule of thumb, a database containing $32k$ models will incur an almost $k$-fold increase in the above processing time requirements (the number of processing elements is assumed fixed and equal to $8K$). Approximately linear speedup can be achieved on a larger Connection Machine (see section 3.8).

In all our experiments, the true model/basis combination was discovered as the pair with the largest weighted vote, i.e., evidence. However, even if the correct model were not found as the maximum winner, it is assumed that a postprocessing stage would be used to verify a number of possible matches. For example, with our database of thirty-two models, there are 7680 possible model/basis combinations. If the first nine or so matching model/basis combinations from the hashing algorithm are checked, we have still achieved a considerable speedup over the alternative of checking all possible matchings. Accordingly, the fact that the accumulated evidence for the ninth model/basis combination is considerably less than the evidence for the winning model/basis combination indicates that the method is robust.

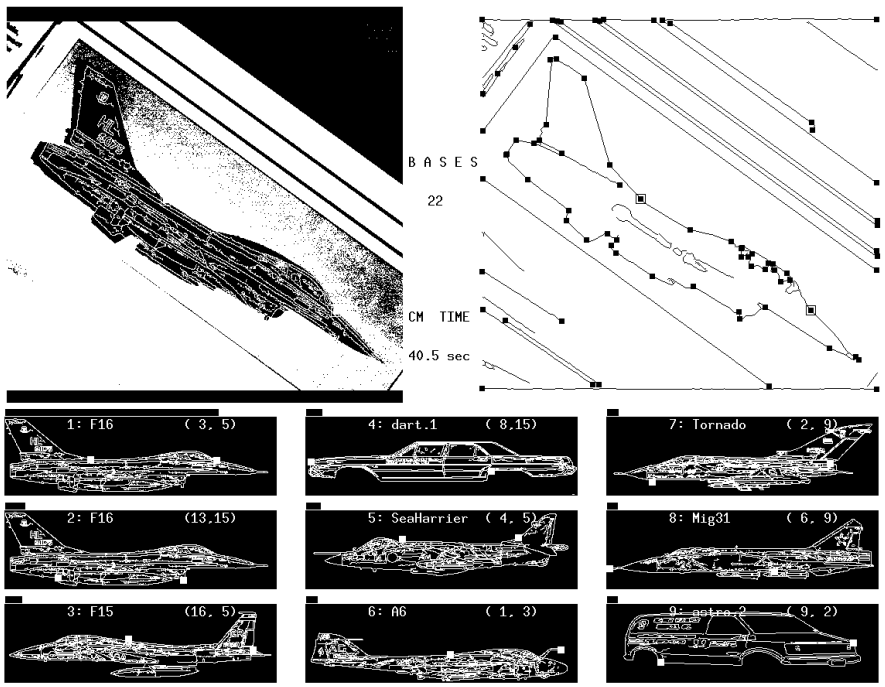


**Figure 8.9** The output of the implementation of our system on the Connection Machine. The test input (F-16) is shown on the top left. The edge map together with the automatically extracted point features is shown on the top right; the basis selection that led to recognition is also marked. A total of 22 basis selections was required, and the elapsed time was 40.5 seconds (NB. this figure does not include the edge detection and feature extraction stages). The bars above each of the 9 top retrieved models provide a length encoding of the total accumulated evidence for the corresponding model/basis combination. The retrieved database model appropriately scaled, rotated and translated is shown overlaid on the test input.

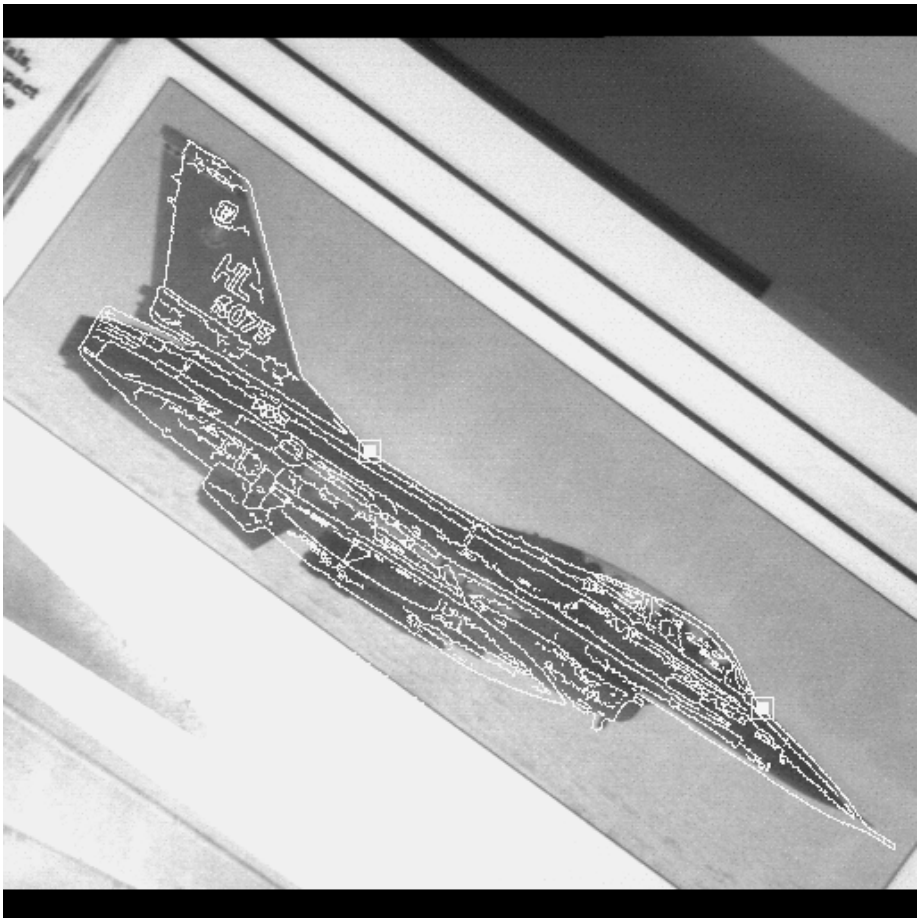

**Figure 8.10 The F16 test input with the retrieved model overlaid on it. The recovered transformation (rotation, translation and scaling) was based solely on the basis pair, and not on a best least-squares match of all corresponding feature pairs.**

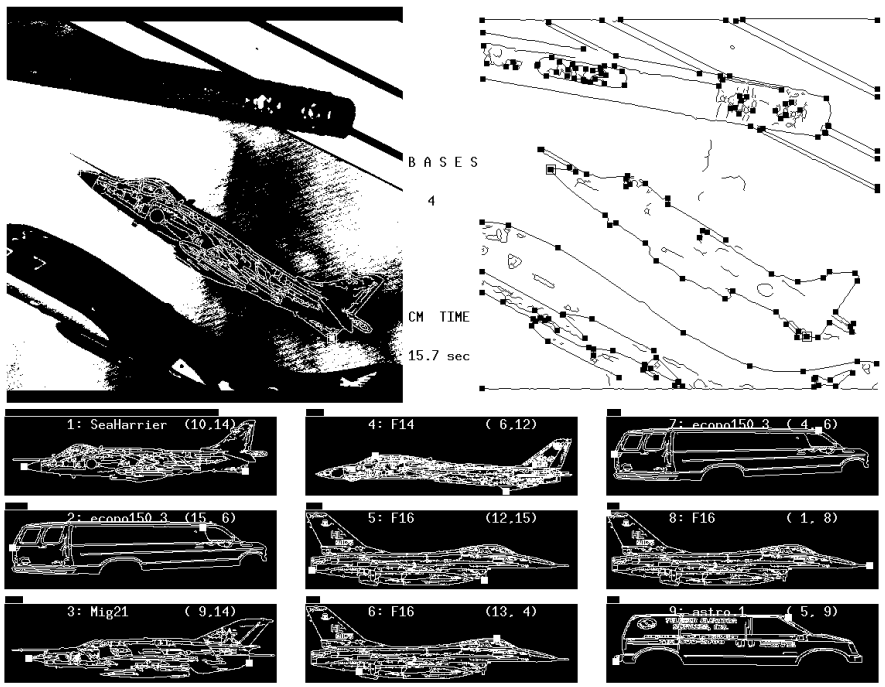


**Figure 8.11** The output of the implementation of our system on the Connection Machine. The test input (Sea Harrier) is shown on the top left. The edge map together with the automatically extracted point features is shown on the top right; the basis selection that led to recognition is also marked. A total of 4 basis selections was required, and the elapsed time was 15.7 seconds (NB. this figure does not include the edge detection and feature extraction). The bars above each of the 9 top retrieved models provide a length encoding of the total accumulated evidence for the corresponding model/basis combination. The retrieved database model appropriately scaled, rotated and translated is shown overlaid on the test input.

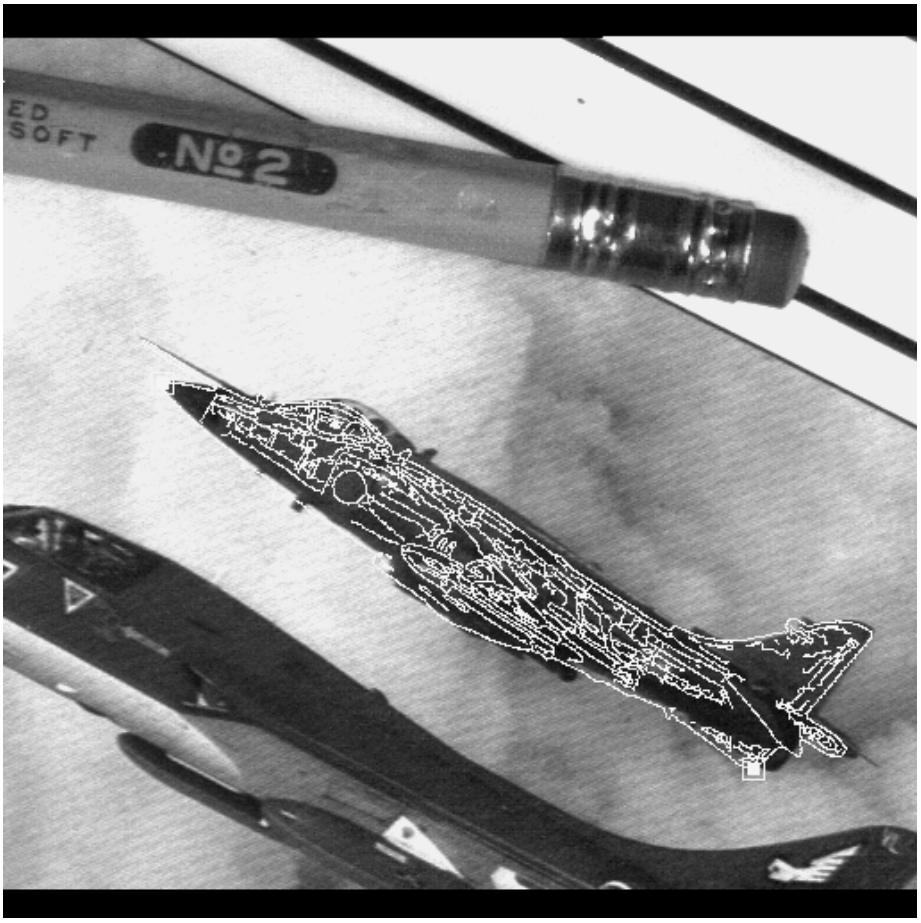


**Figure 8.12** The Sea Harrier test input with the retrieved model overlaid on it. The recovered transformation (rotation, translation and scaling) was based solely on the basis pair, and not on a best least-squares match of all corresponding feature pairs.

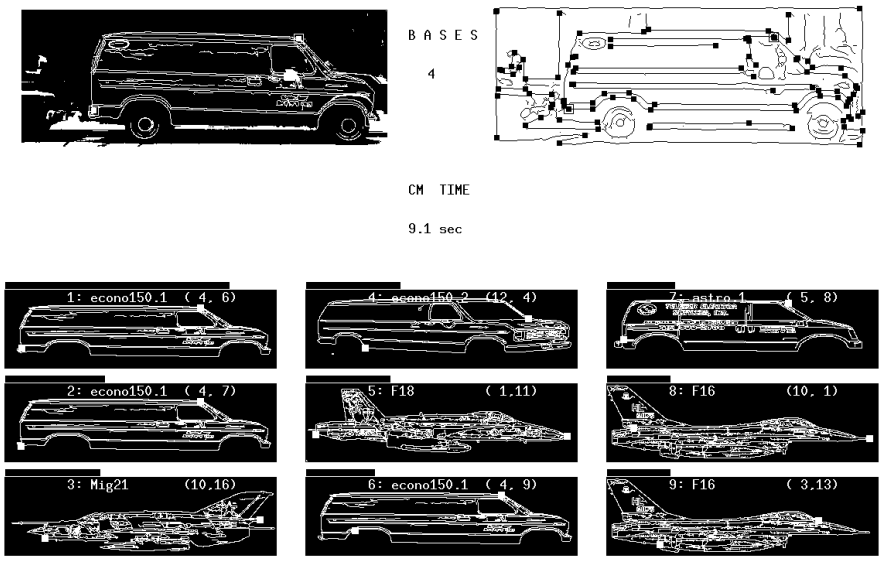


**Figure 8.13** The output of the implementation of our system on the Connection Machine. The test input (Ford *Econoline150*) is shown on the top left. The edge map together with the automatically extracted point features is shown on the top right; the basis selection that led to recognition is also marked. A total of 4 basis selections was required, and the elapsed time was 9.1 seconds (NB. this figure does not include the edge detection and feature extraction). The bars above each of the 9 top retrieved models provide a length encoding of the total accumulated evidence for the corresponding model/basis combination. The retrieved database model appropriately scaled, rotated and translated is shown overlaid on the test input.

**Figure 8.14 The Ford *Econoline 150* test input with the retrieved model overlaid on it. The recovered transformation (rotation, translation and scaling) was based solely on the basis pair, and not on a best least-squares match of all corresponding feature pairs.**

# Chapter 9

# Conclusion

## 9.1 Summary of Results

This dissertation makes three principal contributions. First, the exploitation of parallelism in object recognition is advanced. Two parallel algorithms that realize the geometric hashing method are presented. One algorithm is designed for an SIMD hypercube-based machine; the other algorithm is more general, and relies on data broadcasting capabilities.

The first of the two algorithms is data parallel over the hash table entries and regards geometric hashing as a connectionist algorithm with information flowing via patterns of communication. The second algorithm is inspired by the method of inverse indexing for data retrieval and treats the parallel architecture as a source of "intelligent memory." The algorithm is data parallel over combinations of small subsets of model features.

Per probe of a candidate basis, and using $M\binom{n}{c}(n-c)c!$ processors, the first of the algorithms has time complexity $\mathcal{O}(\log(SMn)\log(Mn))$, whereas the second has time complexity $\mathcal{O}(S+\log(Mn))$; $M$ is the number of database models, $n$ is the number of point features per model, $c$ is the cardinality of the basis tuple, and

$S$ is the number of extracted scene features. The model of parallel computation is the concurrent-read-exclusive-write (CREW) SIMD Hypercube.

The implementations of these two algorithms on a Connection Machine allow the rapid recognition of models consisting of patterns of points, embedded in scenes of several hundred points, independent of rotation, translation or scale changes, and using databases of thousands of models. With $1,024$ synthetic models each consisting of 16 point features, and scenes with 200 point features, we achieve probe times of about 250 milliseconds on a $32K$–processor $CM$-2.

A number of enhancements to the geometric hashing method (such as hash table equalization, and the use of hash table symmetries), which were developed specifically for the parallel algorithms, are also presented. These techniques lead to substantial performance improvements and are also applicable to more general implementations of indexing-based object recognition methods.

A second contribution of this dissertation is an analysis of the expected distribution of computed invariants over the space of invariants, and a related noise sensitivity analysis. In particular, formulas for the expected distributions of computed invariants over the hash space are derived for the cases of rigid, similarity and affine transformations, and for two different distributions (Gaussian and Uniform over a disc) of point features in the model database. For the noise analysis, formulas that describe the dependency of the values for the computed invariants on Gaussian positional error are derived for the similarity and affine transformation cases. The basic underlying assumption here is that the positional accuracy of the extracted features, which is subject to sensor noise and errors introduced during the feature extraction stage, can be modeled by a Gaussian process.

Finally, the third and most important contribution of this dissertation is an

interpretation of geometric hashing that allows the algorithm to be viewed as a Bayesian approach to model-based object recognition. This interpretation, which is a new form of Bayesian-based model matching, leads to well-justified formulas, and gives a precise weighted-voting method for the evidence-gathering phase of geometric hashing. These formulas replace traditional heuristically-derived methods for performing weighted voting, and also provide a precise method for evaluating uncertainty.

A prototype object recognition system has been built using the above ideas, and is implemented on a $CM$-2 Connection Machine. The system is scalable and can recognize models subjected to $2D$ rotation, translation, and scale changes in digital imagery. The models for the object database were obtained from readily available sketches of military aircraft and production automobiles. Unlike military aircraft, which tend to be elongated with distinguishing marks, automobiles have more rotationally symmetric shapes making the recognition task potentially more difficult. Point features based on curvature extrema and singularities of extracted curves were used to build the database. The test inputs to the system are real-world, black and white photographs of airplanes in flight, and of street scenes. The source of the test imagery is distinct from that of the model images.

The object recognition system with enhancements and Bayesian reasoning is then used to locate model objects in the scenes, currently using an $8K$–processor Connection Machine. We obtain extremely good results with similarity-invariant model matching. Currently, the model database consists of 32 objects, and the scenes typically contain a little over a hundred extracted point features.

This system is the first system of its kind that is scalable, uses large databases, can handle noisy input data, works rapidly on an existing parallel architecture,

and exhibits excellent performance with real world, natural scenes.

## 9.2 Future Research Directions

We end this dissertation with a brief mention of possible future research directions.

Currently, in the context of Bayesian geometric hashing and for a given basis selection, the model/basis combination accumulating the largest evidence is retained or rejected based on empirically determined thresholds. An analysis in the spirit of [37] will allow the determination of adaptive thresholds that are a function of the complexities of the viewed scene and the stored models.

Another topic to be explored is the intelligent grouping of features. Currently, a straightforward randomized algorithm selects candidate basis tuples in turn, until recognition is achieved. Methods for grouping image features, as well as their realization in a parallel setting, remain largely unexplored and are expected to greatly expedite the search.

Finally, the use of higher-level image features (i.e. features other than points) for performing object recognition in the context of Bayesian geometric hashing remains largely unexplored. This will require extending the technique and designing evaluation criteria for measuring the relative merit of the different feature types. Related to this topic, is the issue of the development of a control mechanism that will permit the recognition stage of the system to selectively guide the extraction of various features as the recognition process progresses.

# Appendix A

# Some Details Regarding the Derivation of Eqn. 6.6

If we set $\mathcal{X}_1 = (X_1 - x_1)/\sigma$, $\mathcal{X}_2 = (X_2 - x_2)/\sigma$, $\mathcal{Y}_1 = (Y_1 - y_1)/\sigma$, and $\mathcal{Y}_2 = (Y_2 - y_2)/\sigma$, we have that

$$f(X_1, Y_1)dX_1dY_1 = \frac{1}{2\pi}\exp\left(-\frac{\mathcal{X}_1^2 + \mathcal{Y}_1^2}{2}\right)d\mathcal{X}_1d\mathcal{Y}_1 \tag{A.1}$$

$$f(X_2, Y_2)dX_2dY_2 = \frac{1}{2\pi}\exp\left(-\frac{\mathcal{X}_2^2 + \mathcal{Y}_2^2}{2}\right)d\mathcal{X}_2d\mathcal{Y}_2 \tag{A.2}$$

If we set $\mathcal{X} = (X - x)/\sigma$ and $\mathcal{Y} = (Y - y)/\sigma$ we have that

$$f(X, Y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{\mathcal{X}^2 + \mathcal{Y}^2}{2}\right) \tag{A.3}$$

Observing now that

$$X_i = \sigma\mathcal{X}_i + x_i, \quad i = 1, 2 \tag{A.4}$$

and

$$Y_i = \sigma\mathcal{Y}_i + y_i, \quad i = 1, 2 \tag{A.5}$$

we can write

$$\mid J \mid^{-1} = \left[(\sigma\mathcal{X}_2 + x_2 - \sigma\mathcal{X}_1 - x_1)^2 + (\sigma\mathcal{Y}_2 + y_2 - \sigma\mathcal{Y}_1 - y_1)^2\right]. \tag{A.6}$$

In order to express $X$ and $Y$ as functions of $U$ and $V$, we solve Eqns. 6.3 and 6.4 for $X$ and $Y$; then using Eqns. A.4 and A.5, we can show that

$$\mathcal{X} = \mathcal{Z} + \mathcal{C} \tag{A.7}$$

$$\mathcal{Y} = \mathcal{W} + \mathcal{D} \tag{A.8}$$

where

$$\mathcal{Z} = U(\mathcal{X}_2 - \mathcal{X}_1) - V(\mathcal{Y}_2 - \mathcal{Y}_1) + \frac{\mathcal{X}_1 + \mathcal{X}_2}{2} \tag{A.9}$$

$$\mathcal{C} = \frac{1}{\sigma}\left[(U-u)(x_2 - x_1) - (V-v)(y_2 - y_1)\right] \tag{A.10}$$

$$\mathcal{W} = U(\mathcal{Y}_2 - \mathcal{Y}_1) + V(\mathcal{X}_2 - \mathcal{X}_1) + \frac{\mathcal{Y}_1 + \mathcal{Y}_2}{2} \tag{A.11}$$

$$\mathcal{D} = \frac{1}{\sigma}\left[(U-u)(y_2 - y_1) + (V-v)(x_2 - x_1)\right] . \tag{A.12}$$

Eqn. 6.5 can then be rewritten as

$$\begin{aligned} f(U,V) = \frac{1}{8\pi^3\sigma^2} \int_{\mathbb{R}^4} & \exp\left(-\frac{\left((\mathcal{Z}+\mathcal{C})^2 + (\mathcal{W}+\mathcal{D})^2\right)^2}{2}\right) \\ & \cdot \exp\left(-\frac{(\mathcal{X}_1 + \mathcal{Y}_1)^2}{2}\right) \cdot \exp\left(-\frac{(\mathcal{X}_2 + \mathcal{Y}_2)^2}{2}\right) \\ & \cdot \Big[(\sigma\mathcal{X}_2 + x_2 - \sigma\mathcal{X}_1 - x_1)^2 + \\ & \quad (\sigma\mathcal{Y}_2 + y_2 - \sigma\mathcal{Y}_1 - y_1)^2\Big]\, d\mathcal{X}_1 d\mathcal{X}_2 d\mathcal{Y}_1 d\mathcal{Y}_2 . \end{aligned} \tag{A.13}$$

Substitution of expressions A.9 through A.12 in A.13, and integration of the result yields the expression in Eqn. 6.6. □

# Bibliography


[1] **Aho, A. and J. Hopcroft and J. Ullman**. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1984.

[2] **Attneave, F.** Some Informational Aspects of Visual Perception. *Psychological Review*, 61, 1954.

[3] **Ayache, N. and O. Faugeras**. HYPER: A New Approach for the Recognition and Positioning of Two-dimensional Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(1):44–54, 1986.

[4] **Baker, H.**, editor. *The Collected Mathematical Papers of J.J. Sylvester*. Cambridge University Press, 1904-12.

[5] **Ballard, D.** Generalizing the Hough Transform to Detect Arbitrary Shapes. *Pattern Recognition*, 13(2):111–122, 1981.

[6] **Ballard, D.** Parameter Nets: A Theory of Low Level Vision. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 1068–1078, 1981.

[7] **Ballard, D. and C. Brown**. *Computer Vision*. Prentice-Hall, 1982.

[8] **Ballard, D. and D. Sabbah**. Viewer Independent Object Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(6):653–660, 1983.

[9] **Bederson, B., R. Wallace and E. Schwartz**. A Miniaturized Active Vision System. Technical Report 588, Courant Institute of Mathematical Sciences, New York University, October 1991.

[10] **Bergevin, R. and M. Levine**. Generic Object Recognition: Building Coarse $3D$ Descriptions from Line Drawings. In *Proceedings of the IEEE Workshop on Interpretation of $3D$ Scenes*, pages 68–74, Austin, Texas, November 1989.

[11] **Besl, P. and R. Jain**. Three-dimensional Object Recognition. *Computer Surveys*, 17(1):75–145, March 1985.

[12] **Biederman, I.** Human Image Understanding: Recent Research and a Theory. *Computer Vision, Graphics, and Image Processing*, 32:29–73, 1985.

[13] **Boie, R. and I. Cox**. Two Dimensional Optimum Edge Recognition using Matched and Weiner Filters for Machine Vision. In *Proceedings of the International Conference on Computer Vision*, London, England, December 1987.

[14] **Bolle, R., A. Califano, and R. Kjeldsen**. A Complete and Extendable Approach to Visual Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14:534–548, May 1992.

[15] **Bolles, R. and P. Horaud**. 3DPO: A Three-dimensional Part Orientation System. *The International Journal of Robotics Research*, 5:3–26, 1986.

[16] **Bolles, R. and R. Cain**. Recognizing and Locating Partially Visible Objects: the Local Feature Focus Method. *The International Journal of Robotics Research*, 1:57–82, 1982.

[17] **Bourdon, O. and G. Medioni**. Object Recognition Using Geometric Hashing on the Connection Machine. In *Proceedings of the International Conference on Patern Recognition*, pages 596–600, Atlantic City, New Jersey, June 1990.

[18] **Breuel, T.** Model Based Recognition Using Pruned Correspondence Search. In *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference*, Maui, Hawaii, June 1991.

[19] **Brooks, R.** Symbolic Reasoning Around $3D$ Models and $2D$ Images. *Artificial Intelligence*, 17:285–348, 1981.

[20] **Califano, A.** Feature Recognition Using Correlated Information Contained in Multiple Neighborhoods. In *Proceedings of the Seventh AAAI*, pages 831–836, 1987.

[21] **Califano, A. and R. Mohan**. Multidimensional Indexing for Recognizing Visual Shapes. In *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference*, Maui, Hawaii, June 1991.

[22] **Carrihill, B. and R. Hummel**. Experiments with the Intensity Ratio Depth Sensors. *Computer Vision, Graphics, and Image Processing*, 32:337–358, 1985.

[23] **Chen, C. and A. Kak**. A Robot Vision System for Recognizing $3D$ Objects in Low-order Polynomial Time. *IEEE Transactions on Systems Man and Cybernetics*, 19(6):1535–1563, 1989.

[24] **Clemens, D. and D. Jacobs**. Model Group Indexing for Recognition. In *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference*, Maui, Hawaii, June 1991.

[25] **Costa, M., R. Haralick and L. Shapiro**. Optimal Affine Matching. In *Proceedings of the 6th Israeli Conference on Artificial Intelligence and Computer Vision*, Tel Aviv, Israel, December 1989.

[26] **Dickinson, S., A. Pentland, and A. Rosenfeld**. $3D$ Shape Recovery Using Distributed Aspect Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):174–198, 1992.

[27] **Duda, R. and P. Hart**. *Pattern Classification and Scene Analysis*. John Wiley and Sons, 1973.

[28] **Ettinger, G.** Large Hierarchical Object Recognition Using Libraries of Parameterized Model Subparts. In *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference*, Ann Arbor, Michigan, June 1988.

[29] **Fischer, D., R. Nussinov, and H. Wolfson**. $3D$ Substructure Matching in Protein Molecules. In *Proceedings of the International Conference on Pattern Matching*, Arizona, June 1992.

[30] **Fischler, M. and R. Bolles**. Random Sample Consensus: A Paradigm to Model-fitting with Application to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[31] **Flynn, P. and A. Jain**. BONSAI: $3D$ Object Recognition Using Constrained Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(10):1066–1075, 1991.

[32] **Flynn, P. and A. Jain**. $3D$ Object Recognition Using Invariant Feature Indexing of Interpretation Tables. *Computer Vision, Graphics, and Image Processing: Image Understanding*, 55(2):119–129, 1992.

[33] **Forsyth, D., J. Mundy, A. Zisserman, C.Coelho, A. Heller, and C. Rothwell**. Invariant Descriptors for $3D$ Object Recognition and Pose. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(10):971–991, 1991.

[34] **Gavrila, D. and F. Groen**. $3D$ Object Recognition from $2D$ Images Using Geometric Hashing. *Pattern Recognition Letters*, 13(4):263–278, 1992.

[35] **Goad, C.** Special Purpose Automatic Programming for $3D$ Model-based Vision. In *Proceedings of the DARPA Image Understanding Workshop*, 1983.

[36] **Grimson, W. and D. Huttenlocher**. On the Sensitivity of Geometric Hashing. In *Proceedings of the International Conference on Computer Vision*, Osaka, December 1990.

[37] **Grimson, W. and D. Huttenlocher**. On the Verification of Hypothesized Matches in Model-based Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(12):1201–1213, 1991.

[38] **Grimson, W. and T. Lozano-Perez**. Model-based Recognition and Localization from Sparse Range or Tactile Data. *The International Journal of Robotics Research*, 3(3):3–35, 1984.

[39] **Grimson, W. and T. Lozano-Perez**. Localizing Overlapping Parts by Searching the Interpretation Tree. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(4):469–482, 1987.

[40] **Guéziec, A. and N. Ayache**. Smoothing and Matching of $3D$ Space Curves. In *Proceedings of the European Conference on Computer Vision*, Santa Margherita Ligure, Italy, May 1992.

[41] **Hillis, W. and G. Steele**. Data Parallel Algorithms. *Communications of the ACM*, 29, 1986.

[42] **Hong, J. and H. Wolfson**. An Improved Model-based Matching Method Using Footprints. In *Proceedings of the International Conference on Patern Recognition*, Rome, Italy, November 1988.

[43] **Horn, B.** *Robot Vision.* The MIT Press, 1986.

[44] **Hough, P.** Method and Means for Recognizing Complex Patterns, 1962. U.S. Patent 3,069,654.

[45] **Hummel, R. and H. Wolfson**. Affine Invariant Matching. In *Proceedings of the DARPA Image Understanding Workshop*, April 1988.

[46] **Humphreys, G. and V. Bruce**. *Visual Cognition: Computational, Experimental and Neuropsycological Perspectives.* Laurence Erlbaum Associates, 1989.

[47] **Huttenlocher, D.** Three-dimensional Recognition of Solid Objects from a Two-Dimensional Image. Technical Report 1045, Massachussetts Institute of Technology, 1988.

[48] **Huttenlocher, D.** Feature Matching for Object Localization in the Presence of Uncertainty. Technical Report 1133, Massachussetts Institute of Technology, 1990.

[49] **Huttenlocher, D.** Fast Affine Point Matching: An Output-sensitive Method. In *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference*, Maui, Hawaii, June 1991.

[50] **Huttenlocher, D. and S. Ullman**. Recognizing Solid Objects by Alignment. In *Proceedings of the DARPA Image Understanding Workshop*, April 1988.

[51] **Jacobs, D.** Optimal Matching of Planar Models in 3D Scenes. In *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference*, Maui, Hawaii, June 1991.

[52] **Kalvin, A., E. Schonberg, J. Schwartz, and M. Sharir**. Two-dimensional Model-based Boundary Matching Using Footprints. *The International Journal of Robotics Research*, 5(4):38–55, 1986.

[53] **Kim, W.-Y. and A. Kak**. $3D$ Object Recognition Using Bipartite Matching Embedded in Discrete Relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(3):224–251, 1991.

[54] **Kishon, E. and H. Wolfson**. $3D$ Curve Matching. In *Proceedings of the AAAI Workshop on Spatial Reasoning and, Multisensor Fusion*, pages 250–261, St. Charles, Illinois, October 1992.

[55] **Knudsen, E., S. du Lac and S. Esterly**. Computational Maps in the Brain. *Annual Review of NeuroScience*, 10:41–65, 1987.

[56] **Knuth, D.** *Sorting and Searching.* Addison-Wesley, 1973.

[57] **Kriegman, D. and J. Ponce**. Recognizing and Positioning Curved $3D$ Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(12):1127–1137, 1990.

[58] **Lamdan, Y.** *Geometric Hashing.* PhD thesis, New York University, June 1989.

[59] **Lamdan, Y. and H. Wolfson**. Geometric Hashing: A General and Efficient Model-based Recognition Scheme. In *Proceedings of the International Conference on Computer Vision*, pages 238–249, 1988.

[60] **Lamdan, Y. and H. Wolfson**. On the Error Analysis of Geometric Hashing. In *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference*, Maui, Hawaii, June 1991.

[61] **Lamdan, Y., J. Schwartz and H. Wolfson**. Object Recognition by Affine Invariant Matching. In *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference*, pages 335–344, Ann Arbor, Michigan, June 1988.

[62] **Lamdan Y., J. Schwartz, and H. Wolfson**. On Recognition of $3D$ Objects from $2D$ Images. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1407–1413, Philadelphia, PA, April 1988.

[63] **Levine, M.** *Vision in Man and Machine.* McGraw-Hill, 1985.

[64] **Linnainmaa, S, D. Harwood, and L. Davis**. Pose Determination of a Three-dimensional Object Using Triangle Pairs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10:634–647, 1988.

[65] **Lin, W-M. and V. Prasanna Kumar**. Efficient Histograming on Hypercube SIMD Machines. *Computer Vision, Graphics, and Image Processing*, 49:104–120, 1990.

[66] **Little, J., G. E. Blelloch, and T. A. Cass**. Algorithmic Techniques for Computer Vision on a Fine-Grained Parallel Machine. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(3), March 1989.

[67] **Lowe, D.** The Viewpoint Consistency Constraint. *International Journal of Computer Vision*, 1:57–72, 1987.

[68] **Lowe, D. G.** *Perceptual Organization and Visual Recognition.* Kluwer Academic Publishers, 1985.

[69] **Mundy, J. and D. Thompson**. Model-directed Object Recognition on the Connection Machine. In *Proceedings of the DARPA Image Understanding Workshop*, pages 98–104, 1987.

[70] **Mundy, J. and D. Thompson**. Three-dimensional Model Matching from an Unconstrained Viewpoint. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 208–220, Raleigh, NC 1987.

[71] **Narayanan, P. and L. Davis**. Replicated-data Algorithms in Image Processing. Technical Report 536, Center for Automation Research, University of Maryland, College Park, 1991.

[72] **Narayanan, P., L. Chen and L. Davis**. Effective Use of SIMD Parallelism in Low- and Intermediate-level Vision. *IEEE Computer: Special Issue on Parallel Processing for Computer Vision and Image Understanding*, February 1992.

[73] **Nassimi, D. and S. Sahni**. Data Broadcasting in SIMD Computers. *IEEE Transactions on Computers*, C-30:101–107, 1981.

[74] **Preparata, F. and M. Shamos**. *Computational Geometry.* Springer-Verlag, New York, 1985.

[75] **Press, W. H., B. Flannery, S. A. Teukolsky and W. T. Vetterling**. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, 1988.

[76] **Quinn, M. J.** *Designing Efficient Algorithms for Parallel Computers.* McGraw-Hill, 1987.

[77] **Richardson, D.** *The World's Major Military Jets.* Salamander Books Ltd., 1990.

[78] **Rigoutsos, I. and R. Hummel**. Implementation of Geometric Hashing on the Connection Machine. In *Proceedings of the IEEE Workshop on Directions in Automated CAD-Based Vision*, Maui, Hawaii, June 1991.

[79] **Rigoutsos, I. and R. Hummel**. On a Scalable Parallel Implementation of Geometric Hashing on the Connection Machine. Technical Report 554, Courant Institute of Mathematical Sciences, New York University, April 1991.

[80] **Rigoutsos, I. and R. Hummel**. Robust Similarity Invariant Matching in the Presence of Noise. In *Proceedings of the 8th Israeli Conference on Artificial Intelligence and Computer Vision*, Tel Aviv, Israel, December 1991.

[81] **Rigoutsos, I. and R. Hummel**. Massively Parallel Model Matching: Geometric Hashing on the Connection Machine. *IEEE Computer: Special Issue on Parallel Processing for Computer Vision and Image Understanding*, February 1992.

[82] **Roberts, L.** *Optical and Electro-optical Information Processing*, chapter title: Machine Perception of Three-dimensional Solids. MIT Press, Cambridge, Massachusetts, 1965.

[83] **Schreiber, I. and M. Ben-Bassat**. Polygonal Object Recognition. In *Proceedings of the International Conference on Patern Recognition*, pages 852–859, Atlantic City, New Jersey, June 1990.

[84] **Schwartz, J. and H. Wolfson**. Improved Shape-signature and Matching Methods for Model-based Robotic Vision. In *Proceedings of the Workshop on Space Telerobotics*, pages 103–109, JPL, NASA, January 1987.

[85] **Shankar, R., G. Ramamoorthy and M. Suk**. Three-dimensional Object Recognition on the Connection Machine. *Pattern Recognition Letters*, 11(6):485–492, 1990.

[86] **Sossa, H. and R. Horaud**. Model Indexing: the Graph-hashing Approach. In *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference*, Urbana-Champaign, Illinois, June 1992.

[87] **Stanfill, C. and B. Kahle**. Parallel Free Text Search on the Connection Machine System. *Communications of the ACM*, December 1986.

[88] **Stark, L. and K. Bowyer**. Achieving Generalized Object Recognition through Reasoning about Association of Function to Structure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(10):1097–1104, 1991.

[89] **Stein, F. and G. Medioni**. Efficient Two-dimensional Object Recognition. In *Proceedings of the International Conference on Patern Recognition*, pages 596–600, Atlantic City, NewJersey, June 1990.

[90] **Stein, F. and G. Medioni**. Structural Hashing: Efficient Three-dimensional Object Recognition. In *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference*, pages 244–250, Maui, Hawaii, June 1991.

[91] **Stockman, G.** Object Recognition and Localization via Pose Clustering. *Computer Vision, Graphics, and Image Processing*, 40:361–387, 1987.

[92] **Strat, T. and M. Fischler**. Context-based Vision: Recognizing Objects Using Information from both $2D$ and $3D$ Imagery. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(10):1050–1065, 1991.

[93] **Swain, M.** *Color Indexing.* PhD thesis, University of Rochester, 1990.

[94] **Sweetman W. and R. Bonds**. *The Great Book of Modern Warplanes.* Portland House, 1987.

[95] **Thinking Machines**. Parallel Instruction Set: Reference Manual. Technical report, Thinking Machines Corp., 1989.

[96] **Tremblay, M. and D. Poussart**. MAR: An Early Vision System with Integrated Optics and Processing. In *Proceedings of the Canadian Conference on Very Large Scale Integration*, pages 33–40, Vancouver, British Columbia, October 1989.

[97] **Ullman, S.** Aligning Pictorial Descriptions: An Approach to Object Recognition. *Cognition*, 32(3):193–254, 1989.

[98] **Ullman, S. and R. Basri**. Recognition by Linear Combination of Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(10):992–1006, 1991.

[99] **Vayda, A. and A. Kak**. A Robot Vision System for Recognition of Generic Shaped Objects. *Computer Vision, Graphics, and Image Processing: Image Understanding*, 54(1):1–46, 1991.

[100] **Wallace, R., P.-W. Ong, B. Bederson, and E. Schwartz**. Space-variant Image Processing. Technical Report 589, Courant Institute of Mathematical Sciences, New York University, October 1991.

[101] **Wolfson, H., E. Schonberg, A. Kalvin and Y. Lamdan**. Solving Jigsaw Puzzles by Computer Vision. *Annals of Operations Research*, 12:51–64, 1988.